INDEX VAN DE FUNCTIES UIT DE I.B.M.-APL-CURSUS:

)LOAD	*LESSON2*
)COPY	
)FNS	
)VARS	
)WSID	
)SAVE	
)DROP	
)CLEAR	
)LIB	
)ERASE	
←+-×÷*⌈⌊|⍟ *(DYADIC)*	*LESSON3*
+-×÷⌈⌊|! *(MONADIC)*	*LESSON5*
⍴, *(MONADIC)*	*LESSON7*
<≤=≥>≠∨∧~ *(DYADIC)*	*LESSON8*
⍱⍲ *(DYADIC)*	*LESSON9*
/\	*LESSON10*
⍴⍳↑↓ *(DYADIC)*	*LESSON11*
⍳ *(MONADIC)*	
VECTOR INDEXING	
○	*LESSO12M*
FUNCTION DEFENITION	*LESSON13*
CONVERSATIONAL FUNCTIONS	*LESSON14*
⍞⍠; ⍎	
→ *(BRANCHING)*	*LESSON15*
COMPRESSION	
INDEXING	
⌶ (⌶26)	
LABELS	
FUNCTION EDITING	*LESSON16*
?, *(MONADIC)*	*LESSON17*
?,∈⊥⊤ *(DYADIC)*	
⌽⍉⊖⍋⍒ *(MONADIC)*	*LESSON18*
⌽⍉⊖⌿ *(DYADIC)*	

BURROUGHS:

Password: L/10613549 35 [APL]

user nr. sponsor

```
)WIDTH N                                      LESSON19
)DIGITS N                                        |
)ORIGIN N                                        |
)OPR |MESSAGE|                                   |
)PORTS                                           |
)MSG X |MESSAGE|                                 |
)OPRN |MESSAGE|                                  |
)MSGN X |MESSAGE|                                |
)GROUP |.........                                |
)GRPS                                            |
)GRP GPNAME                                      |
)SI                                              |
)SIV                                             |
)PCOPY                                           |
)CONTINUE                                       _|_

∘.(FUNCTION) (UITPRODUCT)                     LESSON20
(FUNCTION).(FUNCTION) (INPRODUCT)               _|_

SPECIALE FUNCTIES:                            LESSON21
   BRANCHING                                     |
   COMMON LIBRARIES                              |
   DATA TYPE CONVERSION                          |
   SCANNING INPUT-STREAMS                        |
   CAI PROGRAMMING TECHNICS                      |
   CORE SAVING TECHNICS                          |
   I-BEAM FUNCTIONS                              |
   ERROR MESSAGES                                |
   TRACING                                       |
   STOP CONTROL                                 _|_

CIRCULAR FUNCTIONS                            LESSO22M
```

-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-o-

IBM.

- Machine aanzetten
- modem aan
- attn.
- nadat keyboard vrij is gegeven" intikken:
  LOGON ⊳ DCS THE (CR) (DCSTHE)
- ENTER PASSWORD
  DCEMTAHT ← AGULSPJH
- les:
) LOAD ⊳ 46 ⊳ LESSON1
- uitzetten:
) OFF
- modem uit
  modem keyboard geblokkeerd is alarm
- apparaat uitschakelen.

02975-77555
-77711
tst 3700
Voor.

)LOAD 46 LESSON1
SAVED 18.14.30 10/06/75

LESSON 1

THIS IS THE FIRST OF 23 MODULES DESIGNED TO TEACH
APL\370 PROGRAMMING.

ALL MODULES BEGIN BY EXECUTING: START
AND ALL CONTAIN A "DESCRIBE" VARIABLE.

START

APL\370 - THE SYSTEM/370 IMPLEMENTATION OF A PROGRAMMING LANGUAGE.

GOOD AFTERNOON!

THIS FIRST LESSON HAS THREE PURPOSES:

1. TO TELL YOU WHAT THE COURSE IS ABOUT.
2. TO INTRODUCE YOU TO APL\370.
3. TO LET YOU PRACTICE USING A TERMINAL.

THIS COURSE IS DESIGNED TO TEACH YOU HOW TO USE APL.
HOWEVER, IT CANNOT TEACH YOU EVERYTHING ABOUT IT, SO
YOU ARE ENCOURAGED TO READ WHATEVER YOU CAN OBTAIN
IN THE WAY OF PUBLICATIONS ON THE SUBJECT; TO ASK
QUESTIONS OF ANYONE MORE EXPERIENCED; AND TO
EXPERIMENT WITH APL\370.

THERE ARE 23 LESSONS IN ALL - 21 TUTORIAL, AND 2 TESTS.
SOME OF THE LESSONS ARE PRIMARILY OF INTEREST TO THE
MATHEMATICS STUDENT, AND ARE INDICATED BY AN "M" SUFFIX
TO THE LESSON NUMBER. THE LESSONS ARE NUMBERED SEQUENTIALLY,
AND SHOULD BE TAKEN IN ORDER.

APL\370 IS A MARRIAGE BETWEEN A PROGRAMMING LANGUAGE (APL) AND A
COMPUTING SYSTEM (IBM SYSTEM/370).

APL IS THE LANGUAGE YOU WILL LEARN TO USE IN THIS COURSE.

THE COMPUTER YOU'RE USING IS ALSO SERVING 0
OTHER USERS AT THIS TIME.

PLEASE TYPE IN YOUR NAME - I DON'T CARE HOW
YOU PUNCTUATE OR SPELL IT:

WHMJ

THANK YOU.

YOUR NAME IS "WHMJ" AND YOUR USER ID
(SIGN-ON NUMBER) IS "0".

WHAT YOU JUST DID WAS TO COMMUNICATE WITH A

COMPUTER - SPECIFICALLY: ONE WHICH IS USING APL\370!

YOU'VE BEEN USING THIS SYSTEM NOW FOR 108.593 *SECONDS.*

HOWEVER, DURING THAT PERIOD, YOU'VE USED ONLY 0.117 *SECONDS OF COMPUTER TIME (SOMETIMES CALLED "CPU TIME").*

THE REASON FOR THE DIFFERENCE IS THAT THE RESOURCES OF THE COMPUTER ARE BEING SHARED AMONG THE 1 *USERS WHO ARE ON NOW.*

FOR THIS REASON, APL\370 IS CALLED A "TIME-SHARING" SYSTEM.

AN APL SYSTEM HAS A VERY SIMPLE STRUCTURE FROM A USER'S VIEWPOINT - ACTIVELY WHEN INTERFACING WITH IT OCCURS IN WHAT IS CALLED AN

ACTIVE WORKSPACE

WHICH IS NOTHING MORE THAN A PART OF THE COMPUTER'S MEMORY SET ASIDE FOR YOUR USE.

WE SHALL NOW USE THE ACTIVE WORKSPACE ("WORKSPACE" IS USUALLY SHORTE to"ws") *AS A DESK CALCULATOR. SUPPOSE WE WANTED THE SUM OF TWO NUMBERS, SA* 5+23, *ALL WE'D HAVE TO ENTER WOULD BE:*

```
      5+23
```

AND THE SYSTEM WOULD RESPOND WITH THE EVALUATION THUS:

```
28
```

THIS IS VALID FOR OTHER ARITHMETIC FUNCTIONS, SUCH AS ×, ÷, *AND* -.

IN APL, SYMBOLS LIKE THESE ARE CALLED:

PRIMITIVE FUNCTIONS

IT IS IMPORTANT TO REMEMBER THAT WHILE 3+5=5+3=8 *OR* 3×5=5×3=15 *(I.E., THE ORDER DOESN'T MATTER),* 3÷5 *IS DIFFERENT FROM* 5÷3, *AND* 3-5 *DIFFERENT RESULT FROM* 5-3. *THE RULE HERE IS:*

89-34 *MEANS SUBTRACT* 34 *FROM* 89, *YIELDING* 55

AND: 72÷8 *MEANS DIVIDE* 72 *BY* 8, *YIELDING* 9

HAVE YOU GOT THAT?

YES

BEFORE YOU TRY WRITING SOME APL EXPRESSIONS, I'D LIKE TO DISCUSS APL'S EVALUATION OF ARITHMETIC EXPRESSIONS INVOLVING 0 *(ZERO), SINCE THIS MAY BE DIFFERENT FROM WHAT YOU LEARNED IN SCHOOL.*

IF "N" IS ANY VALUE EXCEPT 0, *THEN:*

$0-N = -N$
$N-0 = N$
$0+N = N$
$N+0 = N$
$\times$ =

```
      0÷N = 0

      N÷0 = (AN ERROR MESSAGE - THIS IS NOT ALLOWED IN APL)

NOW FOR YOUR PROBLEMS.....

I'M GOING TO PRINT SIMPLE ARITHMETIC STATEMENTS IN PLAIN
ENGLISH, LIKE THIS:

      "THE PRODUCT OF 4 AND 6"

AND I WANT YOU TO ENTER  AN APL EXPRESSION WHICH MEANS WHAT
I PRINTED,  FOROEXAMPLE:

THE QUOTIENT OF 85 AND 3.67
⎕:
      85÷3.67

YOU MAY ENTER ANY OF THESE WHEN YOU SEE THE "⎕" SYMBOL:

      1.  AN APL EXPRESSION REPRESENTING MY STATEMENT
      2.  HELP  --  IF YOU CAN'T DO IT
      3.  STOP  --  IF YOU'VE HAD ENOUGH, AND WE'LL
                    CONTINUE WITH INSTRUCTION.

LET'S GO.....


THE DIFFERENCE OF 53 AND 26

⎕:
      53-26

CORRECT!

THE SUM OF 10 AND 4

⎕:
      10+4

GOOD!

THE PRODUCT OF 7 AND 9

⎕:
      7×9

RIGHT!

THE QUOTIENT OF 4 AND 2

⎕:
      4÷2

CORRECT!

THE DIFFERENCE OF 256 AND 39

⎕:
      STOP
1-03-
HAD ENOUGH, EH?  OK, THEN,
LET'S CONTINUE WITH INSTRUCTION.
```

THIS TYPE OF EXPRESSION IS CALLED A

DYAD

BECAUSE IT HAS TWO (2) ARGUMENTS (THE DATA) AND ONE (1) FUNCTION (OPERATION).

HERE'S AN EXAMPLE OF AN ARITHMETIC DYAD:

5+23

ENTER THE FUNCTION IN THE ABOVE EXPRESSION:

+

CORRECT!

LET' DISCUSS TYPING ERRORS.....

EVERYONE MAKES MISTAKES FROM TIME TO TIME, ESPECIALLY WHEN USING A TERMINAL WITH WHAT MAY BE AN UNFAMILIAR KEYBOARD LAYOUT (APL, FOR EXAMPLE).

IT FOLLOWS, THEN, THAT THERE MUST BE SOME ERROR CORRECTION PROCEDURES AVAILABLE, AND, INDEED, THERE ARE.

HOWEVER, THEY WORK ONLY IF YOU REALIZE YOU'VE MADE AN ERROR BEFORE YOU HIT THE RETURN KEY; ONCE IT'S BEEN PRESSED, IT'S TOO LATE TO PREVENT WHAT YOU TYPED IN BEING TRANSMITTED TO THE COMPUTER.

ALSO, THESE PROCEDURES WILL DIFFER ACCORDING TO THE TYPE OF TERMINAL IN USE (1050, 2741, 3767, COMMUNICATING MAGNETIC CARD TYPEWRITER, SCREEN, ETC.).
WOULD YOU PLEASE ENTER THE TYPE NUMBER OF THE TERMINAL YOU ARE USING (E.G., 3767):
□:

2741

THANKS.

WHEN YOU REALIZE YOU'VE MADE AN ERROR:

1. *BACKSPACE TO THE POINT OF ERROR*
2. *PRESS THE KEY MARKED: ATTN*
3. *TYPE THE REMAINDER OF THE LINE CORRECTLY.*

WHAT HAPPENS IS THAT APL KEEPS TRACK OF WHERE YOU ARE IN THE LINE, AND WHEN YOU INDICATE THAT YOU WISH TO CORRECT SOME OF THE INPUT, IT AUTOMATICALLY ERASES FROM MEMORY EVERYTHING TO THE RIGHT OF, AND INCLUDING, THE POINT OF ERROR.

WITH YOUR HELP, I'LL SIMULATE HOW THIS WORKS.
I'M GOING TO TYPE THE SENTENCE:

1-04- *THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG.*

BACKSPACE TO THE ERROR, AND THEN - THAT'S WHERE YOU COME IN!

WHENEVER YOU SEE THE TYPE HEAD WIGGLE LIKE THIS:

MOVE THE PAPER UP ONE LINE.

OK! HERE GOES.....

THE QUICK BROWN FOX GUMPED OV
v
JUMPED OVER THE LAZY DOG.

IF YOU ROLLED THE PAPER UP AT THE CORRECT TIME, THE RESULT SHOULD LOOK LIKE THIS:

THE QUICK BROWN FOX GUMPED OV
v
JMPED OVER THE LAZY DOG.

LET'S TRY IT ONCE AGAIN:

THE QUICK BROWN FOX JOPPED
v
UMPED OVER THE LAZY DOG.

NOW YOU PRACTICE IT FOR AWHILE.

YOU TYPE -- BUT MAKE SOME ERRORS AND CORRECT THEM. I'LL TYPE BACK TO YOU WHAT I RECEIVE HERE ON THE OTHER END OF THE LINE. TYPE:

READY

WHEN YOU WANT TO CONTINUE WITH INSTRUCTION.

DIT IS WEL EEN STUMP
v
OMPZINNIGE TAST
v
EST!

I SEE: DIT IS WEL EEN STOMPZINNIGE TEST!
READY

I'M FLEXIBLE IN THE MANNER IN WHICH I'LL ACCEPT DATA.

IF YOU WANTED, YOU COULD ENTER:

```
3+7×8÷2
```

LIKE THIS:

```
3+7×8÷2
```

TYPE: READY

WHEN YOU WANT TO CONTINUE WITH INSTRUCTION.

```
4+5×3-8
```
I SEE: 4+5×3-8
READY

SO, WE CAN DEDUCE THIS GENERAL RULE:

WHAT YOU SEE IS WHAT YOU SEND!

THERE ARE CERTAIN CONDITIONS UNDER WHICH EXECUTION OF THE PROGRAMMED COURSE OF INSTRUCTION MAY BE INTERRUPTED --

1--*'PUBLIC ADDRESS' MESSAGES FROM THE APL OPERATOR*
2--*'STATIC' OR NOISE ON THE TELEPHONE LINES*
3--*PREVIOUSLY UNDETECTED 'BUGS' IN THE PROGRAM PRESENTING THE COURSE.*

YOUR REACTION TO ALL THREE INTERRUPTIONS IS CONSISTENT; HOWEVER, IN CASE THREE, RECOVERY CAN NOT BE GUARANTEED. IN CASE YOU ARE NOT SUCCESSFUL IN RECOVERING, CALL YOUR ADVISOR.

IN EACH CASE, INTERRUPTION WILL BE INDICATED BY A MESSAGE, FOLLOWED BY AN INDICATOR.

FOR EXAMPLE, A PUBLIC ADDRESS INTERRUPTION YIELDS THIS:

PA!: APL WILL BE UP TOMORROW.
FNA[23]

THE SECOND LINE IS THE ONE IMPORTANT FOR RECOVERY. IN THIS EXAMPLE, TO RECOVER YOU TYPE: →23.

IN GENERAL, YOU SHOULD TYPE →N (YOU SHOULD SUBSTITUTE FOR N WHATEVER NUMBER APPEARS IN THE SQUARE BRACKETS).

OTHER INTERRUPTION TYPES LOOK LIKE:

(ERROR MESSAGE)
PGMB[86]

OR JUST:

PFC[10]

LET'S TRY IT A FEW TIMES. I'LL SIMULATE AN INTERRUPTION; YOU RECOVER. TYPE: READY
WHEN YOU WISH TO CONTINUE WITH INSTRUCTION.
RANK ERROR
FN[1]
→1

GOOD!

HERE'S ANOTHER:

RANK ERROR

→1

GOOD!

HERE'S ANOTHER:

SYNTAX ERROR
RESUME[5]
RAE
  ∨

 ∨
 EADY

AS YOU SAW EARLIER, APL\370 CAN BE USED AS A DESK CALCULATOR.
HOWEVER, THERE IS FAR MORE TO APL THAN JUST THAT!

RECALL THAT EVERYTHING YOU DO WHEN YOU INTERFACE WITH AN APL SYSTEM
OCCURS IN YOUR _____ WS.

WHAT IS THE MISSING WORD IN THE ABOVE SENTENCE?
ACTIVE
CORRECT!

IT IS CONCEIVABLE THAT WE MIGHT RUN OUT OF SPACE IN AN ACTIVE WORKSPA
OR, THAT WE MIGHT WISH TO SAVE SOME PROGRAMS (CALLED "FUNCTIONS" IN A
OR SOME DATA (CALLED "VARIABLES") FOR LATER USE. THUS, WE SHALL NEE
SOME KIND OF STORAGE FACILITY IN OUR SYSTEM.

APL DOES, INDEED, PROVIDE US WITH A STORAGE FACILITY!
IT IS CALLED A LIBRARY BECAUSE ITS OPERATION IS ANALOGOUS TO THAT OF
A LIBRARY OF BOOKS.

IN AN APL SYSTEM, THE SYSTEM ITSELF IS THE LIBRARIAN, WORKSPACES NOT
IN USE ARE STORED IN ITS LIBRARY, AND THE "BOOK" YOU HAVE AT THE MOME
IS YOUR ACTIVE WS.
SUBSCRIBERS ARE ASSIGNED A QUOTA OF LIBRARY WORKSPACES, AND, IN
WE SHALL ASSUME THAT OUR QUOTA IS 2.

IN ADDITION TO A QUOTA, EACH SUBSCRIBER HAS A SPECIAL WS CALLED
WHOSE NAME CANNOT BE CHANGED. IT HAS SOME SPECIAL USES, AS WE SHALL
DISCUSS LATER.

NOW, IF WE HAVE A LIBRARY, WE MUST HAVE A MEANS OF ACCESSING AND STOR
THINGS IN IT. TO ACCOMPLISH THIS, APL HAS SEVERAL SYSTEM COMMANDS, W
ARE EASY TO DISTINGUISH, BECAUSE THEY ALL START WITH:)
(A RIGHT PARENTHESIS).

IN THE NEXT MODULE, WE SHALL BE DISCUSSING SOME OF THESE SYSTEM COMMA
AND HERE IS A LIST OF THOSE WE SHALL LOOK AT, AND WHAT THEY DO:

COMMAND	PURPOSE
)LOAD	LOADS A WS FROM A LIBRARY INTO YOUR ACTIVE WS.
)COPY	COPIES ALL (OR PART) OF A LIBRARY WS INTO YOUR ACTIV
)FNS	LISTS ALL THE FUNCTIONS IN YOUR ACTIVE WS.
)VARS	LISTS ALL THE VARIABLES IN YOUR ACTIVE WS.
)WSID	DISPLAYS OR CHANGES THE NAME OF YOUR ACTIVE WS.
)OFF	SIGNS YOU OFF THE SYSTEM.

)OFF HOLD SIGNS YOU OFF, BUT HOLDS THE LINE OPEN FOR 60 SECONDS
SO THAT YOU CAN SIGN ON AGAIN WITH A DIFFERENT NUMBER.
)SAVE STORES THE ACTIVE WS INTO A LIBRARY.
)DROP DISCARDS A WS FROM A LIBRARY.
)CONTINUE CHANGES THE NAME OF YOUR ACTIVE WS TO "CONTINUE",
SAVES IT, THEN SIGNS YOU OFF THE SYSTEM.
)CLEAR CLEARS YOUR ACTIVE WS OF ALL FUNCTIONS AND VARIABLES

AND THAT'S IT FOR THE FIRST LESSON!

WHEN YOU'RE READY FOR LESSON 2, SIGN ON, THEN ENTER:

)LOAD 46 LESSON2

AND THE SYSTEM WILL RESPOND WITH THE TIME AND DATE IT WAS LAST SAVED.
(IF YOU GET AN ERROR MESSAGE, CHECK YOUR TYPING, THEN TRY AGAIN.)

ONCE IT'S LOADED (AND THE TIME AND DATE MESSAGE SHOWS THAT IT IS, IN
NOW IN YOUR ACTIVE WS), ENTER:

)START

IMPORTANT

AFTER YOU SIGN OFF, REMEMBER TO SWITCH OFF THE TERMINAL.

ALSO, IF YOU HAVE A TYPE 113 DATA SET, YOU MUST PRESS
THE "TALK" BUTTON AFTER SWITCHING OFF THE TERMINAL, OR
THE TELEPHONE LINE WILL REMAIN CONNECTED, THUS TYING
UP ONE ACCESS PORT WHICH SOMEONE ELSE COULD BE USING.

'BYE!

)LOAD 46 LESSON2
SAVED 14.57.33 10/14/75

LESSON 2

THIS IS THE FIRST OF TWO LESSONS
DEALING WITH SYSTEM COMMANDS.

JUST TYPE: START

TO BEGIN THE LESSON.

START

NOTE

IF, FOR ANY REASON, IT SHOULD BECOME NECESSARY FOR YOU TO ABORT A MODULE, THIS CAN BE DONE WHENEVER THE SYSTEM IS AWAITING YOUR INPUT.

USE EITHER (1) OR (2) BELOW AS APPROPRIATE.

1. IF THE SYMBOLS: ⍞: ARE PRINTED AT THE LEFT MARGIN, THEN ENTER A RIGHT ARROW: → FOLLOWED BY "RETURN".

2. IF THOSE SYMBOLS ARE NOT PRINTED, ENTER THIS SEQUENCE: "O", BACKSPACE, "U", BACKSPACE, "T" (I.E., THE LETTERS: O U T OVERSTRUCK) FOLLOWED BY "RETURN".

SUCH ACTION WILL CAUSE AN IMMEDIATE EXIT FROM THE PROGRAMMED COURSE OF INSTRUCTION, AFTER WHICH YOU MAY SIGN OFF.

LESSON 2

THIS IS THE FIRST OF TWO LESSONS
DEALING WITH SYSTEM COMMANDS.

IN AN APLSV SYSTEM (I.E., NOT APL/CMS, FOR EXAMPLE)
THE SIGN-ON PROCEDURE FOLLOWING LINE CONNECTION IS TO
ENTER A RIGHT PARENTHESIS FOLLOWED BY A USER IDENTITY -
GENERALLY A MAN-NUMBER.

THIS, IN TURN, MAY BE OPTIONALLY FOLLOWED BY A COLON AND
A SECURITY PASSWORD.

FOR EXAMPLE:

)123456

OR

)123456:*LOCK*

THE NUMERIC PORTION OF THE ENTRY IS FREQUENTLY REFERRED TO AS A "SIGN-ON NUMBER".

PLEASE TYPE IN YOUR SIGN-ON NUMBER - NO RIGHT PARENTHESIS OR LOCK - JUST THE NUMBER ITSELF:

(IF YOU'RE NOT ON AN APLSV SYSTEM, USE ANY 6-DIGIT, POSITIVE INTEGER.)

987654

NOW ENTER YOUR NAME - 2 INITIALS AND LAST NAME SUGGESTED:

WHMJ

THANK YOU.

LET'S REVIEW WHAT HAPPENS WHEN YOU SIGN ON TO AN APL SYSTEM.

ASSUME THAT YOU'VE JUST DIALED IN TO AN APLSV SYSTEM -

NOW SIGN ON:

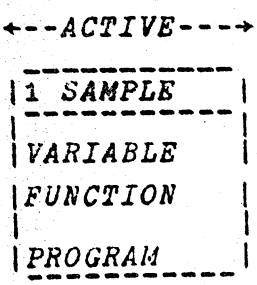
(AND, FROM NOW ON, IF YOU CAN'T ANSWER ANY QUESTION IN THIS LESSON, JUST TYPE: HELP)

987654
RIGHT PARENTHESIS MISSING.
INCORRECT USER ID.
)987654
018) 14.35.51 9/16/76 *WHMJ*

. A P L . S V .

GOOD!

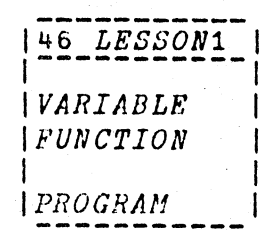
WHEN YOU SIGN ON, THE SYSTEM AUTOMATICALLY ASSIGNS YOU A FRESH, CLEAN WORKSPACE AS YOUR ACTIVE WORKSPACE, AND YOU MIGHT PICTURE IT AS THIS:



OK. ASSUME YOU ARE JUST STARTING THE COURSE.
YOU HAVE JUST SIGNED ON - NOW LOAD THE FIRST LESSON INTO YOUR ACTIVE WS:

```
      )LOAD 46 LESSON1
SAVED 15.42.15 9/19/75
```

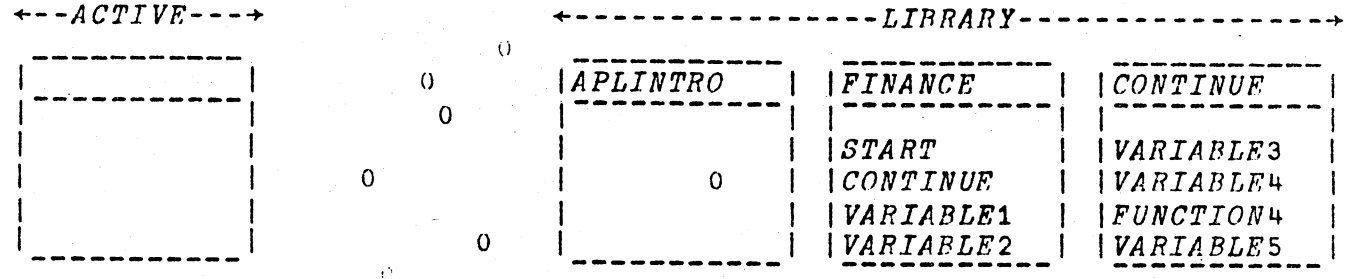
WHAT HAPPENED WAS THAT A COPY OF A WS CALLED "46 LESSON1" WAS LOADED INTO YOUR ACTIVE WS, WHICH NOW LOOKS LIKE:



GOOD.
NOW LET'S TRY A FEW EXERCISES. I'LL GIVE YOU SOME THINGS TO DO, YOU DO THEM, THEN I'LL DIAGNOSE WHAT YOU DID. ALSO, I'LL DISPLAY THE STATE OF YOUR ACTIVE WS AND LIBRARY EACH TIME.

USE THE COMMAND ")CLEAR" TO CLEAR YOUR ACTIVE WS:

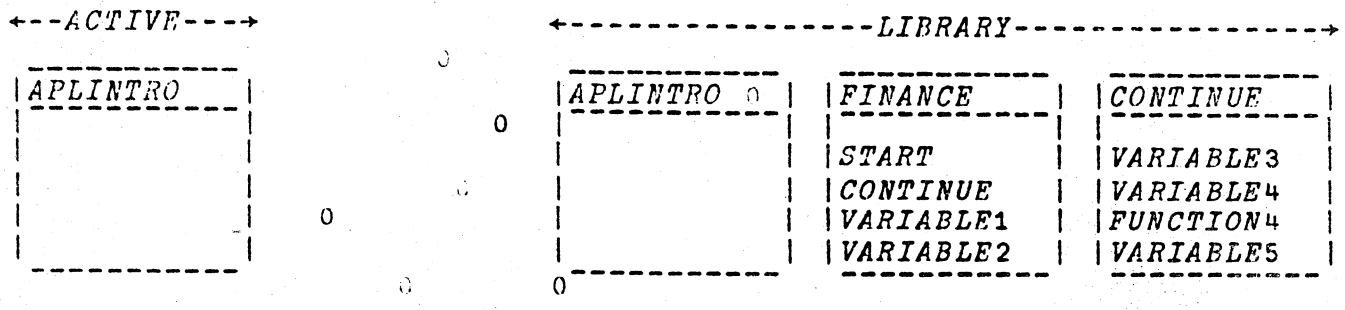
```
      )CLEAR
CLEAR WS
```



NOW - LOAD THE WS NAMED "APLINTRO" FROM YOUR LIBRARY INTO YOUR ACTIVE WORKSPACE:

```
      )LOAD APLINTRO
SAVED 15.42.15 9/19/75
```

HERE'S WHAT IT LOOKS LIKE NOW:



APL WORKSPACE NAMES CAN BE IDENTIFIED OR CHANGED USING THE ')WSID' COMMAND. A PLAIN ')WSID' WILL CAUSE THE PRESENT NAME OF YOUR ACTIVE WORKSPACE TO BE IDENTIFIED. MOREOVER, IF ')WSID' IS FOLLOWED BY THE DESIRED NEW WORKSPACE NAME (AS IN ')WSID NEWNAME'), THEN THE NAME OF YOUR ACTIVE WORKSPACE WILL BE CHANGED TO THE NEW NAME INDICATED.

NOW YOU TRY IT - - -

FIND OUT THE NAME OF YOUR ACTIVE WORKSPACE:

```
     )WSID
APLINTRO
```

NOW CHANGE THE NAME OF THE ACTIVE WORKSPACE TO: FOOTBALL

```
     )WSID FOOTBALL
WAS APLINTRO
```

THE RESPONSE YOU RECEIVED IS THE STANDARD ONE FOR NAME CHANGES, AND WE SHALL NOW USE A DIFFERENT, BUT FASTER, PICTORIAL REPRESENTATION OF YOUR WORKSPACE STATES:

```
WSTYPE    WSNAME      <------------------CONTENTS---------------->

ACTIVE    FOOTBALL
LIBRARY   APLINTRO
LIBRARY   FINANCE     START      CONTINUE   VARIABLE1  VARIABLE2
LIBRARY   CONTINUE    VARIABLE3  VARIABLE4  FUNCTION4  VARIABLE5
```

SUPPOSE YOU WOULD LIKE TO KNOW THE NAMES OF THE WORKSPACES STORED IN YOUR LIBRARY. THE SYSTEM COMMAND ')LIB' WILL GIVE IT TO YOU. EXECUTE IT NOW TO SEE WHAT IT DOES; THEN COMPARE WHAT IT GIVES TO THE LATEST PICTORIAL REPRESENTATION ABOVE.

```
     )LIB
APLINTRO
FINANCE
CONTINUE
```

AND THAT'S WHAT YOU GET!

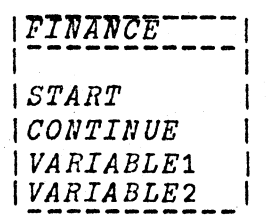
I'D LIKE YOU TO PRACTICE USING THE COMMANDS ')FNS' AND ')VARS' FOR AWHILE. HERE'S WHAT YOU SHOULD DO. LOAD ONE OF THE WORKSPACES IN YOUR LIBRARY INTO YOUR ACTIVE WORKSPACE; THEN USE ')FNS' AND ')VARS' TO SEE WHAT'S THERE. DO THIS AS MANY TIMES AS YOU LIKE WITH AS MANY OF YOUR WORKSPACES AS YOU LIKE. WHEN YOU ARE READY TO CONTINUE WITH INSTRUCTION, JUST TYPE 'READY'.

DON'T BE SURPRISED IF, ON OCCASION, YOU GET WHAT APPEARS TO BE NO RESPONSE TO THESE COMMANDS - THIS MERELY MEANS THAT THERE ARE NO FUNCTIONS (OR VARIABLES) IN THE ACTIVE WS.

(REMEMBER - "')FNS" LISTS FUNCTIONS (PROGRAMS) AND "')VARS" LISTS VARIABLES (DATA)).

```
     )LOAD FINANCE
SAVED 15.42.15 9/19/75
     )VARS
VARIABLE1  VARIABLE2
     )FNS
START      CONTINUE
     READY
```

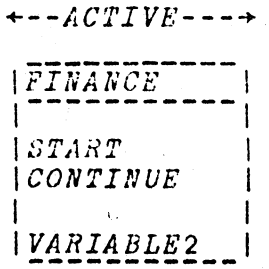
YOU NOT ONLY CAN LIST THE FUNCTIONS AND VARIABLES IN YOUR ACTIVE WORKSPACE, BUT ALSO CAN ERASE ANY OF THEM SELECTIVELY. THE SYSTEM COMMAND TO DO THIS IS ')ERASE OBJECTNAME', WHERE 'OBJECTNAME' IS THE NAME OF THE FUNCTION OR VARIABLE YOU DESIRE TO ERASE. FOR EXAMPLE, LET'S SAY YOUR ACTIVE WORKSPACE NOW LOOKS LIKE THIS:



USE THE ")ERASE" COMMAND TO REMOVE ONE OF THE OBJECTS FROM YOUR ACTIVE WORKSPACE:

```
)ERASE VARIABLE1
```

HERE'S THE RESULTING ACTIVE WORKSPACE:



LET'S LOOK AT THE COPY COMMAND.....

IT HAS SEVERAL FORMATS, WHICH COPY ALL, OR PART, OF A LIBRARY WS INTO YOUR ACTIVE WS, THUS:

```
)COPY WSNAME
)COPY LIBNO WSNAME
)COPY WSNAME OBJECTNAME
)COPY LIBNO WSNAME OBJECTNAME
```

AND "OBJECTNAME" IS THE NAME OF THE DESIRED FUNCTION OR VARIABLE WHICH RESIDES IN THE SPECIFIED LIBRARY WORKSPACE.

LET'S GIVE IT A TRY.....

FIRST: CLEAR YOUR ACTIVE WORKSPACE:

```
)CLEAR
CLEAR WS
```

```
WSTYPE     WSNAME      ←-----------------CONTENTS---------------→

ACTIVE
LIBRARY    APLINTRO    FUNCTION1  FUNCTION2  FUNCTION3
LIBRARY    FINANCE     START      CONTINUE   VARIABLE1  VARIABLE2
LIBRARY    CONTINUE    VARIABLE3  VARIABLE4  FUNCTION4  VARIABLE5
```

O.K. - NOW COPY AN OBJECT INTO YOUR ACTIVE WORKSPACE FROM ONE OF YOUR LIBRARY WORKSPACES:

```
      )FINANCE
      v
      )COPY FINANCE CONTINUE
SAVED 15.44.51 9/13/75

WSTYPE    WSNAME      <------------------CONTENTS--------------->

ACTIVE                CONTINUE
LIBRARY   APLINTRO    FUNCTION1  FUNCTION2  FUNCTION3
LIBRARY   FINANCE     START      oCONTINUE  VARIABLE1  VARIABLE2
LIBRARY   CONTINUE    VARIABLE3  VARIABLE4  FUNCTION4  VARIABLE5
```

THERE ARE TWO MORE SYSTEMS COMMANDS YOU'LL NEED IN ORDER TO FULLY CONTROL YOUR WORKSPACE AND LIBRARIES:

)SAVE *AND:* *)DROP*

YOU MUST BE CAREFUL WHEN USING THESE COMMANDS BECAUSE IT IS EASY TO DISCARD (DROP) A WORKSPACE FROM A LIBRARY AND YOU DIDN'T INTEND TO, OR TO OVERWRITE A WORKSPACE, AGAIN WHEN YOU DIDN'T INTEND TO DO SO.

LET'S EXAMINE THE ')DROP' COMMAND FIRST:

ITS PERMISSIBLE FORMATS:

```
      )DROP WSNAME
```

OR:

```
      )DROP LIBNO WSNAME
```

THIS COMMAND OPERATES ON LIBRARY WORKSPACES ONLY; THAT IS, ITS USE HAS NO EFFECT WHATSOEVER UPON THE ACTIVE WORKSPACE.

NOW - SEE AN EXAMPLE OF ITS USE BY USING THE ')DROP WSNAME' FORMAT TO DISCARD ANY WORKSPACE FROM YOUR PRIVATE LIBRARY.

REFER TO THE LATEST PICTORIAL REPRESENTATION ABOVE, AND DISCARD ANY WORKSPACE EXCEPT "CONTINUE". I'D LIKE YOU TO RETAIN THE CONTENTS OF THAT WS BECAUSE WE'LL USE THEM FOR ANOTHER EXAMPLE LATER IN THIS MODULE.

GO AHEAD:

```
      )DROP FINANCE
  14.51.11  9/16/76
```

AND HERE'S YOUR RESULTING LIBRARY STATUS:

```
WSTYPE    WSNAME      <------------------CONTENTS--------------->

ACTIVE                0 CONTINUE
LIBRARY   APLINTRO    FUNCTION1  FUNCTION2  FUNCTION3
LIBRARY
LIBRARY   CONTINUE    VARIABLE3  VARIABLE4  FUNCTION4  VARIABLE5
```

SO FAR, WE'VE LOOKED AT USING THE ')DROP' COMMAND ON A PRIVATE LIBRARY. IF THIS COMMAND WERE USED ON A WORKSPACE IN A PUBLIC LIBRARY (LIBRARY 46, WHICH CONTAINS THIS COURSE, IS AN EXAMPLE OF A PUBLIC LIBRARY) IT WOULD BE EXECUTED ONLY IF YOU WERE THE USER WHO HAD STORED IT THERE IN THE FIRST PLACE.

FOR EXAMPLE: I STORED (SAVED) A WS CALLED "LESSON1" IN LIBRARY 46. IF YOU ATTEMPTED TO DISCARD IT BY ENTERING:

)DROP 46 LESSON1

YOU WOULD RECEIVE THE FOLLOWING MESSAGE:

IMPROPER LIBRARY REFERENCE

AND NO ACTION WOULD BE TAKEN BY THE SYSTEM.

THIS NOW LEADS US TO THE ')SAVE' COMMAND.

THE FORMAT FOR USING ')SAVE' IS:

)SAVE

OR

)SAVE WSNAME

THE FIRST FORM SAVES YOUR ACTIVE WORKSPACE AUTOMATICALLY UNDER ITS PRESENT NAME. THE SECOND FORM ESSENTIALLY DOES THE SAME THING, BUT ALLOWS YOU TO EXPLICITLY SPECIFY A NAME.

LET'S GO THROUGH A FEW EXERCISES. FIRST, I WILL CLEAR ALL OF YOUR WORKSPACES EXCEPT 'CONTINUE'.

THE STATE OF YOUR WORKSPACES IS NOW:

WSTYPE	WSNAME	←----------CONTENTS----------→				
ACTIVE						
LIBRARY						
LIBRARY						
LIBRARY	CONTINUE	VARIABLE3	VARIABLE4	FUNCTION4	VARIABLE5	

SAVE A COPY OF THE ACTIVE WS BY EXECUTING:)SAVE

```
      )SAVE
NOT SAVED, THIS WS IS CLEAR WS
```

THAT'S O.K. - THAT WAS A TRAP!
A COPY OF THE ACTIVE WORKSPACE WILL NOT BE SAVED UNLESS:

1) THE ACTIVE WORKSPACE ALREADY HAS A NAME (OTHER THAN "CLEAR")

OR:

2) YOU EXPLICITLY NAME IT IN THE ')SAVE' COMMAND BY USING THE THE ')SAVE WSNAME' FORMAT.

AND:

3) YOU HAVE NOT FILLED YOUR WORKSPACE QUOTA.

NOW: NAME AND SAVE THE ACTIVE WS BY EXECUTING:)SAVE FINANCE


```
      )WSID FINANCE
```

```
INCORRECT SYSTEM COMMAND.
     )SAVE FINANCE
  14.54.52  9/16/76

WSTYPE    WSNAME     ←------------------CONTENTS--------------→

ACTIVE    FINANCE
LIBRARY   FINANCE
LIBRARY
LIBRARY   CONTINUE   VARIABLE3  VARIABLE4  FUNCTION4  VARIABLE5

CHANGE THE NAME OF THE ACTIVE WS TO ANYTHING
OTHER THAN "FINANCE" OR "CONTINUE":


     )WSID KAREL
WAS FINANCE

NOW SEE AN EXAMPLE OF SAVING IT BY EXECUTING: )SAVE
     )SAVE
   14.56.8  9/16/76 KAREL

WSTYPE    WSNAME     ←------------------CONTENTS--------------→

ACTIVE    KAREL
LIBRARY   FINANCE
LIBRARY   KAREL
LIBRARY   CONTINUE   VARIABLE3  VARIABLE4  FUNCTION4  VARIABLE5

CHANGE THE NAME AGAIN TO ANYTHING OTHER THAN
"FINANCE", "CONTINUE", OR "KAREL" AND THEN SAVE IT:

     )WSID KEREL
WAS KAREL

GOOD!
HERE'S THE STATE OF YOUR LIBRARY NOW:


WSTYPE    WSNAME     ←------------------CONTENTS--------------→

ACTIVE    KEREL
LIBRARY   FINANCE
LIBRARY   KAREL
LIBRARY   CONTINUE   VARIABLE3  VARIABLE4  FUNCTION4  VARIABLE5

NOW CONTINUE WITH THE PROBLEM (I.E., SAVE THE ACTIVE WS):

     )SAVE
NOT SAVED, WS QUOTA USED UP

ANOTHER TRAP!!!

DO YOU UNDERSTAND THE MESSAGE?

↑ES
YES OR NO?
YR
 ^
 ES

YOU MAY SOMETIMES RECEIVE THE MESSAGE:
```

NOT SAVED, THIS WS IS (WSNAME2)

FOLLOWING A SAVE COMMAND OF THE FORM:

)SAVE (WSNAME1)

THIS IS A FEATURE OF APL DESIGNED TO PROTECT YOU FROM INADVERTENTLY OVERWRITING AN EXISTING WS IN A LIBRARY.

IF YOU HAD INTENDED TO OVERWRITE, USE ')WSID NEWNAME' FOLLOWED BY ')SAVE'.

HENCE, FOR YOU TO SAVE "KEREL" YOU'LL HAVE TO DROP EITHER "FINANCE" OR "KAREL".

ONE OF THE FEATURES WHICH HAS CONTRIBUTED TO THE POPULARITY OF APL\370 IS THE BUILT-IN CAPABILITY TO LOAD, COPY, OR COPY FROM ANY WORKSPACE IN THE SYSTEM, PROVIDED YOU KNOW THE LIBRARY IN WHICH IT RESIDES.

IF, FOR EXAMPLE, YOU WISHED TO LOAD YOUR FRIEND'S WORKSPACE NAMED "PROGRAMS", YOU WOULD HAVE TO KNOW THE NUMBER OF HIS PRIVATE LIBRARY. THIS IS THE NUMERIC PORTION OF HIS USER ID (I.E., HIS SIGN-ON NUMBER). IT IS NOT NECESSARY TO ENTER YOUR PRIVATE LIBRARY NUMBER WHEN ACCESSING WORKSPACES IN YOUR OWN PRIVATE LIBRARY.

IF YOUR FRIEND'S SIGN-ON NUMBER WERE "123456", THEN YOU'D HAVE TO EXECUTE THE COMMAND:

)LOAD 123456 PROGRAMS

TO OBTAIN A COPY OF THE WORKSPACE.

FURTHERMORE, IT IS POSSIBLE TO PROTECT A WS WITH A PASSWORD, IN A SIMILAR MANNER TO THE WAY IN WHICH A USER ID IS PASSWORD-PROTECTED.

FOR EXAMPLE, SUPPOSE YOUR FRIEND HAD LAST SAVED HIS WS CALLED "PROGRAMS" LIKE THIS:

)SAVE PROGRAMS:P25A1

HE WOULD HAVE SET A LOCK ON THE WS WHICH WOULD PREVENT IT FROM BEING ACCESSED BY ANYONE (INCLUDING HIMSELF!) UNLESS THIS SAME PASSWORD WERE INCLUDED WITH THE WORKSPACE NAME WHENEVER THE WS WERE REFERENCED.

IF YOU HAD TRIED TO USE THE FORM OF THE LOAD COMMAND IN THE PREVIOUS EXAMPLE, YOU WOULD HAVE RECEIVED THIS ERROR MESSAGE:

WS LOCKED

AND NO ACTION WOULD HAVE BEEN TAKEN BY THE SYSTEM.

TO REMOVE A PASSWORD, YOUR FRIEND WOULD ENTER:

)SAVE PROGRAMS:

AND YOU USE THE SAME TECHNIQUE TO SET AND REMOVE PASSWORDS FROM YOUR USER ID.

WE MENTIONED EARLIER THAT SOME WORKSPACES RESIDE IN PUBLIC LIBRARIES. THESE LIBRARIES HAVE 1 TO 3 DIGIT NUMBERS AND ARE DESIGNED TO CONTAIN WORKSPACES WHICH HOLD FUNCTIONS OF GENERAL USEFULNESS, ESPECIALLY "CANNED" APPLICATIONS.

ASSUME YOU WANT TO ACCESS SOME WORKSPACE OR OBJECT IN LIBRARY 1.

THE AVAILABLE SYSTEM COMMANDS YOU COULD USE ARE:

)LIB 1 - LISTS ALL WORKSPACES IN LIBRARY 1.

)LOAD 1 WSNAME - LOADS A COPY OF "WSNAME" FROM LIBRARY 1 INTO YOUR ACTIVE WORKSPACE.

)COPY 1 WSNAME - COPIES EVERYTHING FROM "1 WSNAME" INTO YOUR ACTIVE WS, BUT DOES NOT CHANGE THE NAME OF YOUR ACTIVE WS. OBJECTS IN THE ACTIVE WS WITH NAMES IDENTICAL TO THOSE IN "1 WSNAME" WILL BE OVERWRITTEN.

)COPY 1 WSNAME OBJECTNAME - COPIES THE SPECIFIED OBJECT FROM "1 WSNAM INTO YOUR ACTIVE WS, AND OVERWRITES ANY EXISTING OBJECT OF THE SAME NAME.

LET'S SEE IF YOU UNDERSTAND ALL THAT.....

HERE'S A 3-PART PROBLEM.

(DO ONE STEP AT A TIME!)

1. LIST ALL THE WORKSPACES IN LIBRARY 1.

2. LOAD WORKSPACE "SAMPLE" FROM LIBRARY 1 INTO YOUR ACTIVE WS.

3. COPY INTO YOUR ACTIVE WS THE FUNCTION "PROGRAM" FROM WS "APL" IN LIBRARY 1.

OFF YOU GO.....

```
      )LIB 1
SAMPLE
APL
PLOTFORM
WSFNS
```

CORRECT SO FAR - - PLEASE CONTINUE:

```
      )LOAD 1 SAMPLE
SAVED 15.43.25 9/15/75
```

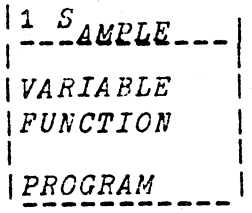
GOOD - NOW DO THE LAST PART:

```
      )COPY 1 APL PROGRAM
SAVED 15.42.15 9/19/75
```

THERE! YOU DID IT!

AND HERE'S YOUR RESULTING ACTIVE WORKSPACE:

←--ACTIVE---→



THIS COMPLETES LESSON 2, THE FIRST OF 2 MODULES ABOUT SYSTEM COMMANDS (THE OTHER IS LESSON 19).

THE NEXT MODULE IS "46 LESSON3" WHICH DISCUSSES SOME BASIC ARITHMETIC OPERATIONS USING PRIMITIVE APL FUNCTIONS.

```
      )LOAD 46 LESSON3
SAVED  18.43.50 10/06/75
```

LESSON 3

THIS MODULE TEACHES THE DYADIC USE OF
THESE PRIMITIVE FUNCTIONS:

```
      +  -  ×  ÷  *  ⌈  ⌊  |  ⍟
```

START

LESSON 3

IN THIS LESSON, WE SHALL LEARN THE DYADIC USE
OF THESE PRIMITIVE FUNCTIONS:

```
      +  -  ×  ÷  *  ⌈  ⌊  |  ⍟
```

AND HERE'S WHAT THE STRANGE-LOOKING ONES DO:

```
      *  ←→  POWER (EXPONENT): 2*4 = 2×2×2×2
      ⌈  ←→  MAXIMUM: 3⌈5 = 5      27⌈20 = 27
      ⌊  ←→  MINIMUM: 3⌊5 = 3      27⌊20 = 20
      ⍟  ←→  LOGARITHM:  10⍟100 = 2     2⍟8 = 3
      |  ←→  RESIDUE (ALGEBRAIC REMAINDER): 2|16 = 0
             1|4.36 = 0.36      3|¯4 = 2
```

LET'S TAKE A CLOSER LOOK AT THE RESIDUE FUNCTION:

IN ITS GENERAL FORM:

```
      A|B
```

IT YIELDS A RESULT, R, SUCH THAT:

```
      R = B-A×Q
```

WHERE:

```
      Q -- IS AN INTEGER
AND:
      R -- IS EITHER:
              1. IF "A" AND "B" ARE POSITIVE, THE
                 "REMAINDER" OF B÷A.
           OR 2. THE SMALLEST INTEGER FOR
                 THE DEFINITION TO BE TRUE (IF EITHER
                 "A" OR "B" IS NEGATIVE).
```

NOW THAT'S PRETTY CONFUSING, SO LET'S WORK OUT A
COUPLE OF EXAMPLES.....

EXAMPLE 1:

```
      3|5
```

THE RESULT IS 2 BECAUSE THE ARGUMENTS ARE BOTH POSITIVE, AND 5÷3 IS 1, WITH 2 REMAINING.

IF WE HAD:

```
      5|3
3
```

WE SEE THAT THE RESULT IS 3, BECAUSE 5 INTO 3 "WON'T GO", WHICH LEAVES A REMAINDER OF 3.

EXAMPLE 2:

```
      ¯3|8
¯1
```

NOW LET'S WORK THIS ONE OUT THE WAY APL WOULD, AND SEE WHY THE RESULT IS ¯1:

Q	*A×Q*	*B-A×Q*	*COMMENTS*
1	¯3	11	*FIRST PASS.....*
2	¯6	14	*"R"'S GETTING LARGER - TRY A NEGATIVE "Q".....*
¯1	3	5	*THAT'S BETTER.....*
¯2	6	¯2	0 *BETTER YET.....*
¯3	9	¯1	*THAT'S THE SMALLEST WE'LL FIND!*

THE BALANCE OF THIS LESSON CONSISTS OF PROBLEMS OF THIS TYPE:

```
      ¯2⌊7
```

AND YOU HAVE TO ENTER WHAT YOU THINK APL'S EVALUATION OF EACH EXPRESSION WILL BE.

IF YOU GET STUCK, TYPE: HELP
AND I'LL TELL YOU THE ANSWER.

YOU ARE PERMITTED ONLY 1 ATTEMPT AT EACH QUESTION, AND THE LESSON WILL END AFTER YOU HAVE TRIED 50 PROBLEMS, OR WHENEVER YOU ENTER: STOP INSTEAD OF AN ANSWER. EITHER WAY, YOU'LL GET A "SCORE" INDICATING YOUR PERCENTAGE OF CORRECT ANSWERS.

ONE LAST POINT BEFORE YOU BEGIN - THERE'S A DIFFERENCE IN APL BETWEEN THE SYMBOLS FOR A VALUE WHICH IS NEGATIVE, AND FOR THE ARITHMETIC OPERATION OF SUBTRACTION:

```
      ¯8 MEANS:  NEGATIVE 8
      -8 MEANS:  SUBTRACT (POSITIVE) 8
```

NOW RE-READ THE FOREGOING (IF NECESSARY), AND, WHENEVER YOU'RE READY TO BEGIN, TYPE: READY

```
       READY

      3|1

1
```

CORRECT!

9

GOOD!

10●1000

3

RIGHT!

1÷8

.125

CORRECT!

1|3

0

GOOD!

-4L7

-4

RIGHT!

1×3

3

CORRECT!

10●10

1

GOOD!

0*4

0

RIGHT!

3*9

HEEL VEEL

ANSWER IS: 19683
TRY THIS ONE:

0|5

5

CORRECT!

2÷8

```
.25
GOOD!
      4|6
2
RIGHT!
      ¯3⌊¯8
¯8
CORRECT!
      10●100
2
GOOD!
      1+2
3
RIGHT!
      ¯1+1
0
CORRECT!
      2-¯2
4
GOOD!
ATTENTION
READY[2]
       →2
      16⍟2
8
RIGHT!
      ¯1⌈4
4
CORRECT!
      1+¯7
¯6
GOOD!
      4|5
1
```

RIGHT!

```
      3+¯6
¯3
```

CORRECT!

```
      0×¯8
0
```

GOOD!

```
      0⌊1
0
```

RIGHT!

```
      4-¯4
8
```

CORRECT!

```
      2|2
0
```

GOOD!

```
      5×¯8
¯40
```

RIGHT!

```
      4⌊¯9
¯9
```

CORRECT!

```
      0+¯2
¯2
```

GOOD!

```
      1÷4
.25
```

RIGHT!

```
      ¯3-10
¯13
```

CORRECT!

```
      2⌈3
```

```
3
GOOD!
      5-2
3
RIGHT!
     ⌊5⌈7
7
CORRECT!
      0×¯4
0
GOOD!
      2*8
256
RIGHT!
      0|4
4
CORRECT!
      ¯3×4
¯12
GOOD!
      1-¯7
≠
∨
8
RIGHT!
      1+¯6
¯5
CORRECT!
      2⌈¯8
2
GOOD!
      ¯3⌈7
7
RIGHT!
```

```
      10*100

2

CORRECT!

      2|9

1

GOOD!

      0⌈6

6

RIGHT!

      2|4

0

CORRECT!

      2|9

1

GOOD!

      2-2

0

RIGHT!

      8÷2

4

CORRECT!

      ¯3-5

¯8

GOOD!

AND THAT'S THE END OF LESSON 3!

YOUR SCORE IS: 100 °/° REPRESENTING
50 CORRECT OUT OF 50 ATTEMPTS.

THE NEXT MODULE WILL BE "46 LESSON4", IN WHICH
WE'LL DISCUSS VARIABLES, SPECIFICATION, AND LITERALS.
```

```
      )LOAD 46 LESSON4
SAVED  18.44.13 10/06/75
```

LESSON 4

THIS LESSON TEACHES VARIABLES,
SPECIFICATION, AND LITERALS.

```
      START
```


LESSON 4

IN THIS MODULE WE SHALL LEARN HOW TO ASSIGN A VALUE TO A VARIABLE.

LET'S BEGIN WITH ASSIGNING NUMERIC VALUES.

GIVEN THE VALUE "2.476" WHICH WE WOULD LIKE TO BE ASSOCIATED WITH THE VARIABLE "K", WE ASSIGN (SOMETIMES CALLED "SPECIFY") THUS:

```
      K←2.476
```

AND APL WILL STORE THE VALUE OF 2.476 IN "K".

NOW FOR THE OTHER SIDE OF THE COIN!

TO OBTAIN THE VALUE OF A VARIABLE, SIMPLY TYPE IN ITS NAME:

```
      K
2.476
```

IF, HOWEVER, THERE WERE NO VARIABLE WITH THAT NAME IN THE ACTIVE WS, YOU WOULD GET AN ERROR MESSAGE.

LET ME SHOW YOU WHAT I MEAN - I'M GOING TO ERASE "K":

```
      )ERASE K
```

AND, NOW THAT IT'S BEEN ERASED, I'LL ASK FOR ITS VALUE:

```
      K
VALUE ERROR
      K
      ^
```

IT IS ALSO POSSIBLE TO STORE A COMPUTED VALUE IN A VARIABLE:

```
      K←3+2×8÷4
```

AND IF WE ASK FOR "K" WE'LL GET ITS VALUE, THUS:

```
      K
7
```

NOW IT'S YOUR TURN!

)1-

SPECIFY THE VARIABLE "V" AS HAVING THE VALUE: 99.12
(IF YOU CAN'T, TYPE: HELP)

V←99.12

GOOD!
LET'S TRY ONCE MORE TO ENSURE YOU UNDERSTAND.

ASSIGN 4+8*3 TO "VAR":

VAR←4+8*3

GREAT!

WE CAN ALSO SPECIFY LITERAL (I.E., NON-NUMERIC) VALUES TO VARIABLES. THIS IS DONE BY ENCLOSING THE VALUE IN SINGLE QUOTES (UPPER CASE "K") LIKE THIS:

```
      Z←'WHAT TIME IS IT, PLEASE?'
```

AND IF WE REQUESTED A DISPLAY OF "Z", WE'D GET:

```
      Z
WHAT TIME IS IT, PLEASE?
```

SOMETIMES YOU MIGHT WANT TO HAVE A SINGLE QUOTE AS PART OF THE VALUE. IN THIS CASE, REPRESENT IT AS A DOUBLE QUOTE, THUS:

```
      M4←'I CAN''T DO IT!'
```

WHICH DISPLAYS AS:

```
      M4
I CAN'T DO IT!
```

IMPORTANT!

YOU MUST ALWAYS HAVE AN EVEN NUMBER OF SINGLE QUOTES WHEN DEALING WITH LITERAL DATA. IF YOU DON'T, THE SYSTEM WILL JUST SIT THERE (FOREVER, IF NECESSARY!) UNTIL THE NUMBER OF QUOTES IS EVEN.

SO, IF YOU'VE ENTERED LITERAL DATA WHICH SEEMS TO BE VALID, BUT NOTHING HAPPENS, AND THE KEYBOARD IS UNLOCKED, TRY ENTERING A SINGLE QUOTE.

(ALSO, IF YOU'RE WONDERING WHERE THE "!" SYMBOL IS ON THE KEYBOARD, DON'T, BECAUSE IT ISN'T. IT'S FORMED BY UPPER CASE "K", THEN BACKSPACE, AND THEN ".".)

NOW YOU TRY SOME:

SPECIFY "Q" TO HAVE THE VALUE: GOOD FOR YOU!

Q←'GOOD FOR YOU!'

TERRIFIC!

ONCE MORE.....

SPECIFY "S" TO BE: DON'T FORGET

(BE CAREFUL!)

S←'DON''T FORGET'

VERY GOOD!

LET'S CONTINUE.....

IT IS QUITE IN ORDER TO HAVE MORE THAN ONE SPECIFICATION IN A SINGLE APL EXPRESSION. FOR INSTANCE, THE FOLLOWING:

```
      W←3.14159×R←(D←25)÷2
```

WHICH APL EVALUATES FROM RIGHT TO LEFT (REMEMBER THAT!) LIKE THIS:

1. SPECIFY "D" TO HAVE THE VALUE 25
2. DIVIDE "D" BY 2, YIELDING THE VALUE 12.5
3. SPECIFY "R" TO BE 12.5 (D÷2)
4. MULTIPLY 3.14159 BY R (12.5), YIELDING 39.269875
5. SPECIFY "W" TO BE 39.269875

AS A GENERAL RULE, YOU CAN SAY THAT THE VALUE ASSIGNED TO A VARIABLE IS THE EVALUATION OF EVERYTHING TO ITS RIGHT, EXCEPT AS MODIFIED BY PARENTHESES, AS THE EXAMPLE ILLUSTRATED.

NOW LET ME SEE YOU DO ONE LIKE THIS.

IN ONE (1) APL EXPRESSION (ONE LINE) I WANT YOU TO:

1. SPECIFY "K" TO HAVE THE VALUE 6
2. SPECIFY "B" TO BE "K" DIVIDED BY 2
3. SPECIFY "A" TO BE 2 TIMES "B"

GO TO IT.....

WHEN YOU'VE FINISHED, TYPE: READY
AND I'LL CHECK IT OUT FOR YOU.

```
      A←2×(B←(K←6)÷2)
READY
```

WELL DONE!

NOW THAT YOU KNOW HOW TO USE THE SPECIFICATION ARROW, LET'S DISCUSS THE CHARACTERS YOU ARE PERMITTED TO USE IN A VARIABLE NAME.

THEY ARE:

1. ANY ALPHABETIC CHARACTER, WITH OR WITHOUT AN UNDERSCORE. (UNDERSCORING A LETTER IS ACCOMPLISHED BY BACKSPACING, THEN ENTERING THE UNDERSCORE CHARACTER (UPPER CASE "F") UNDER THE DESIRED LETTER.)

2. ANY DIGIT (I.E., THE CHARACTERS: 0 1 2 3 4 5 6 7 8 9)

3. THE SYMBOL: Δ (UPPER CASE "H")

THE RULES ARE SIMPLE:

1. A VARIABLE NAME CONSISTS OF FROM 1 TO 77 CHARACTERS, OF WHICH THE FIRST 31 CHARACTERS MUST BE UNIQUE. PRACTICALLY SPEAKING, THIS MEANS 1-31 CHARACTERS.

2. THE FIRST CHARACTER MUST BE ALPHABETIC (WITH OR WITHOUT UNDERSCORE)

HERE ARE SOME EXAMPLES OF VALID VARIABLE NAMES:

```
CAW1Δ0G
Z
```

Y0
F1
PLOT
DDT

AND THIS BRINGS UP AN IMPORTANT POINT - SINCE IT IS VALID TO HAVE A VARIABLE NAME WITH ADJACENT CHARACTERS AND LETTERS, THERE IS NO MEANING IN APL FOR THE CONVENTIONAL MATHEMATICAL NOTATION 3X (MEANING 3×X). IT IS NECESSARY TO STATE, ALWAYS, THE EXPLICIT OPERATION YOU WANT PERFORMED UPON THE DATA.

IF YOU WERE TO ENTER:

```
      4X←2
```

YOU WOULD GET THIS RESPONSE:

```
SYNTAX ERROR
      4 X←2
        ^
```

BECAUSE APL FIRST SPECIFIES "X" TO HAVE THE NUMERIC VALUE OF 2, THEN LOOKS FOR ANOTHER PRIMITIVE FUNCTION TO THE LEFT OF "X", CAN'T FIND ONE, BUT NOTICES ANOTHER VALUE, 4, BUT DOES NOT KNOW WHAT OPERATION YOU WANT PERFORMED ON THE 4 AND THE 2, SO IT STOPS EXECUTION, AND GENERATES THE "SYNTAX ERROR" MESSAGE.

BEAR IN MIND THAT, IN THIS EXAMPLE, YOU WILL HAVE SUCCEEDED ONLY IN SPECIFYING X←2.

HAD WE WANTED 4×X←2 WE SHOULD HAVE ENTERED JUST THAT, AND APL WOULD HAVE EVALUATED THAT TO BE 8.

AND THAT'S IT FOR SPECIFICATION!

THE NEXT MODULE IS "46 LESSON5", WHICH DEALS WITH THE MONADIC USE OF SOME PRIMITIVE FUNCTIONS.

```
      )LOAD 46 LESSON5
SAVED  18.44.55 10/06/75
```

LESSON 5

THIS MODULE DISCUSSES THE MONADIC
USE OF THESE PRIMITIVE FUNCTIONS:

      + - × ÷ ⌈ ⌊ | !

      START

          LESSON 5

IN THIS LESSON WE SHALL LEARN THE MONADIC
USE OF THESE PRIMITIVE FUNCTIONS:

      + - × ÷ ⌈ ⌊ | !

7 OF WHICH YOU HAVE ALREADY USED DYADICALLY.

THE TERM MONADIC MEANS THAT THE FUNCTION
HAS ONLY 1 ARGUMENT, AND SUCH AN EXPRESSION
IS CALLED A MONAD (AS OPPOSED TO A DYAD, WHICH,
YOU WILL RECALL, HAS 2 ARGUMENTS). THE SINGLE
ARGUMENT IS ALWAYS TO THE RIGHT OF THE FUNCTION,
AND IT MAY BE A SINGLE VALUE OR A COMPUTED
VALUE (AS WE'LL SEE NEXT LESSON).

HERE'S WHAT THESE MONADIC FUNCTIONS DO:

SYMBOL	NAME	OPERATION
+	IDENTITY	RETURNS THE ARGUMENT
-	NEGATE	CHANGES THE ARGUMENT'S SIGN
×	SIGNUM	YIELDS: ¯1, 0, OR 1, DEPENDING UPON THE ARGUMENT BEING NEGATIVE, ZERO, OR POSITIVE, RESPECTIVELY
÷	RECIPROCAL	YIELDS: 1÷ARGUMENT
⌈	CEILING	SMALLEST INTEGER GREATER THAN OR EQUAL TO THE ARGUMENT
⌊	FLOOR	LARGEST INTEGER LESS THAN OR EQUAL TO THE ARGUMENT
!	FACTORIAL	YIELDS: 1×2×3×.....×ARGUMENT
|	ABSOLUTE VALUE	RETURNS THE ARGUMENT, BUT WITH ITS SIGN MADE POSITIVE, IF NECESSARY

AND HERE ARE SOME EXAMPLES OF THEM IN USE:

```
      +¯6.7
¯6.7

      -¯3
3

      ×¯9
¯1

      ÷5
```

```
0.2

      ⌈¯3.4
¯3

      ⌊4.6
4

      !3
6

      |¯0.07
0.07
```

THE REST OF THIS LESSON CONSISTS OF EXPRESSIONS SIMILAR TO THE ONES ABOVE, AND YOU HAVE TO ENTER WHAT YOU THINK APL'S EVALUATION OF THEM WILL BE.

YOU MAY ENTER: STOP (INSTEAD OF AN ANSWER) TO TERMINATE THE LESSON; OTHERWISE, IT WILL END AUTOMATICALLY AFTER YOU'VE ANSWERED 50 PROBLEMS.

YOU MAY ENTER: HELP (INSTEAD OF AN ANSWER) IF YOU GET STUCK (BUT THAT WON'T COUNT TOWARDS THE 50!).

AT THE END OF THE LESSON, YOU'LL RECEIVE A "SCORE" SHOWING YOUR PERCENTAGE OF CORRECT ANSWERS.

NOW EXAMINE THE EXAMPLES ABOVE, AND, WHENEVER YOU WOULD LIKE TO BEGIN, TYPE: READY

```
      READY

      ×1.5

1
```

CORRECT!

```
      +0.3

.3
```

GOOD!

```
      |¯3.5

¯3
```

RIGHT!

```
      ⌈5.5

6
```

CORRECT!

```
      ÷8

.125
```

GOOD!

```
      |¯2.9
```

2.9

RIGHT!

|2.7

2.7

CORRECT!

×4.3

1

GOOD!

÷2

.5

RIGHT!

⌈4.3

5

CORRECT!

-0.3

⁻.3

GOOD!

+1.8

1.8

RIGHT!

|0.9

.9

CORRECT!

÷1

1

GOOD!

-⁻1.3

1.3

RIGHT!

⌈⁻1.9

⁻1

CORRECT!

-3.9

```
⁻3.9
GOOD!
      |⁻0.2
.2
RIGHT!
      ⌈2.4
3
CORRECT!
      ÷4
.25
GOOD!
      -5.6
⁻5.6
RIGHT!
      ⌊0.2
0
CORRECT!
      !1
1
GOOD!
      |⁻2.7
2.7
RIGHT!
      ÷1
1
CORRECT!
      -0.4
⁻.4
GOOD!
      -5.4
⁻5.4
RIGHT!
      !1
```

```
1
CORRECT!
      x¯2.9
¯1
GOOD!
      ⌈¯2.2
¯2
RIGHT!
      !4
24
CORRECT!
      !2
2
GOOD!
      ⌊4.5
4
RIGHT!
      ⌈4.7
5
CORRECT!
      ÷4
.25
GOOD!
      !8
40320
RIGHT!
    0 ⌊¯1.5
¯2
CORRECT!
      !2
2
GOOD!
      |3.8
```

```
3.8
RIGHT!
      +3.7
3.7
CORRECT!
      +1.5
1.5
GOOD!
      |0.7
.7
RIGHT!
      !4
24
CORRECT!
      !1
1
GOOD!
      ⌊0.3
0
RIGHT!
      ÷2
.5
CORRECT!
      -¯1.1
1.1
GOOD!
      +2.1
2.1
RIGHT!
      ⌈3.3
4
CORRECT!
      ÷4
```

.25

GOOD!

WHICH BRINGS US TO THE END OF LESSON 5.

YOUR SCORE: 100 °/° (50 *CORRECT OUT OF* 50 *ATTEMPTS*).

THE NEXT MODULE IS "46 *LESSON6T*", *WHICH IS*
A 21-*QUESTION TEST ON BASIC SYSTEM COMMANDS*.

```
      )LOAD 46 LESSON6T
SAVED  18.45.28 10/06/75
```

LESSON 6T

THIS MODULE IS A 21-QUESTION
TEST ON BASIC SYSTEM COMMANDS.

      START

LESSON 6T

BASIC APL SYSTEM COMMANDS QUIZ

FOR EACH QUESTION IN THIS TEST YOU WILL BE GIVEN
TWO CHANCES TO ENTER THE CORRECT ANSWER; FAILURE
TO ANSWER CORRECTLY AFTER TWO ATTEMPTS RESULTS IN
THE PROGRAM CONTINUING ON TO THE NEXT PROBLEM.

YOU WILL RECEIVE A "SCORE" AT THE END OF THE TEST
AS A PERCENTAGE; HOWEVER, THOSE QUESTIONS WHICH
WERE ANSWERED CORRECTLY ON THE FIRST TRY WILL BE
GIVEN TWICE THE WEIGHT OF THOSE ANSWERED CORRECTLY
ON THE SECOND ATTEMPT.

THERE ARE 21 QUESTIONS IN ALL, AND NO ACCOUNT WILL
BE TAKEN OF THE TIME IT TAKES YOU TO ANSWER.

IMPORTANT

IN THIS TEST, ANY QUESTION REQUIRING AN ANSWER
OF "YES" OR "NO" MUST HAVE THESE RESPONSES
SPELLED OUT IN FULL.

GOOD LUCK!

1. ASSUME YOUR IDENTIFICATION NUMBER IS 123456. SIGN ON THE SYSTEM.
```
      )123456
```
CORRECT!

2. LOAD YOUR WORKSPACE NAMED BUGGY.
```
      )LOAD BUGGY
```
CORRECT!

3. LIST ALL FUNCTIONS IN IT.
```
      )FNS
```
CORRECT!

4. LIST THE VARIABLES IN IT.
```
      )VARS
```
CORRECT!

5. COPY IN YOUR ENTIRE WORKSPACE NAMED HELPFUL.

)COPY HELPFUL
CORRECT!

6. COPY IN A FUNCTION FLIT FROM YOUR WORKSPACE SPRAY.
)COPY SPRAY FLIT
CORRECT!

7. COPY IN THE VARIABLE DDT FROM 594159'S WS NAMED DEADLY
)COPY 594159 DEADLY DDT
CORRECT!

8. ERASE THE FUNCTION SWAT.
)ERASE SWAT
CORRECT!

9. DISCARD0YOUR WS NAMED HELPFUL.
)DROP HELPFUL
CORRECT!

10. ASSUMING THAT YOU HAD ACTUALLY EXECUTED ALL OF THE ABOVE COMMANDS
IF YOU THEN EXECUTED ')SAVE', WHAT WOULD THE NAME OF THE SAVED WORKSP/
BUGGY
CORRECT!

11. CAN YOU SAVE THIS WORKSPACE UNDER THE NAME 'BETTER' DIRECTLY?
NO
INCORRECT - TRY AGAIN:
YES
CORRECT!

12. CAN YOU SAVE IT UNDER THE NAME 'CONTINUE' DIRECTLY?
YES
CORRECT!

13. CHANGE THE NAME OF THIS WORKSPACE TO 'BETTER'.
)WSID BETTER
CORRECT!

14. GET THE CURRENT NAME OF THE ACTIVE WORKSPACE.
)WSID
CORRECT!

15. SAVE IT WITHOUT USING ITS NAME.
)SAVE
CORRECT!

16. SAVE IT USING ITS NAME.
)SAVE BETTER
CORRECT!

17. LIST ALL YOUR WORKSPACES.
)LIB
CORRECT!

18. GET A CLEAR WORKSPACE.
)CLEAR
CORRECT!

19. LIST THE WORKSPACES IN LIBRARY 46.
)LIB 46
CORRECT!

20. SIGN OFF THE SYSTEM (WITHOUT DUMPING ACTIVE WORKSPACE IN CONTINUE)
- LEAVE PHONE LINE OPEN.

)OFF HOLD
CORRECT!

21. SIGN OFF SYSTEM (NO DUMP IN CONTINUE).
)OFF
CORRECT!

SCORE: 98 %
AND THAT ENDS THIS FIRST TEST!

THE NEXT MODULE WILL BE "46 LESSON7", WHICH
IS THE FIRST OF THREE WHICH COVER 'VECTORS'.

```
      )LOAD 46 LESSON7
SAVED  18.46.17 10/06/75

LESSON 7

THIS IS THE FIRST OF THREE LESSONS ABOUT
VECTORS -- THE OTHERS ARE LESSONS 10 AND 11.

      START

LESSON 7


COPYRIGHT 1975, IBM CORPORATION.


IN THIS MODULE YOU WILL LEARN WHAT VECTORS ARE,
AND HOW TO SOLVE EVERYDAY PROBLEMS USING THEM.

THIS, FOR EXAMPLE:

29 54 51 11 41 58

IS A VECTOR WITH 6 NUMBERS OR "ELEMENTS",
SEPARATED BY ONE (AT LEAST) OR MORE BLANKS.

HOW MANY ELEMENTS HAS THIS VECTOR?

44 73 87 71 80 70 74
7
THAT'S RIGHT!
THE NUMBER OF ELEMENTS IN A VECTOR IS ALSO ITS LENGTH.

THE EXPRESSION: 44 73 87 71 80 70 74
IS, THEREFORE, A VECTOR WHOSE LENGTH IS 7
ENTER A NUMERIC VECTOR OF LENGTH 2:
16 345


WOULD YOU LIKE TO GIVE ME ANOTHER ONE?
YES

ENTER A NUMERIC VECTOR OF LENGTH 7:
1 2 3 4 5 6 7


WOULD YOU LIKE TO GIVE ME ANOTHER ONE?
YES

ENTER A NUMERIC VECTOR OF LENGTH 4:
4 3 2 1


WOULD YOU LIKE TO GIVE ME ANOTHER ONE?
NO

ASSIGN A (NUMERIC) VECTOR OF LENGTH 4 TO A VARIABLE:
```

```
A←1 2 3 4
```

GOOD - YOUR VECTOR IS NOW STORED IN THE VARIABLE "A".
WOULD YOU LIKE TO ASSIGN ANOTHER ONE?
NO

IF, LATER, YOU WISH TO DETERMINE THE LENGTH OF "A",
THERE IS A (MIXED MONADIC) FUNCTION WHICH YIELDS
ONLY THE LENGTH OF A VECTOR WITHOUT DISPLAYING
ITS CONTENTS. IT IS: ⍴ (UPPER SHIFT "R").

IT IS CALLED "RHO" (PRONOUNCED "ROW" AS IN BOAT).

IF:
```
      A←1 2 3 4
```
THEN:
```
      ⍴A
4
```
A VECTOR WAS ASSIGNED TO THE VARIABLE "ALF",
FIND ITS LENGTH:
```
⍴ALF
5
```
RIGHT!
DISPLAY THE ELEMENTS IN ALF:
```
ALF
49 67 68 20 91
```

GOOD - HOW ABOUT THIS:

SOMETHING WAS ASSIGNED TO THE VARIABLE "X17".
WHAT'S ITS LENGTH?
```
⍴X17
```

THAT'S RIGHT!

FIND OUT WHAT'S IN X17:
```
X17
890
```

A SCALAR!!!

A SINGLE NUMBER (SCALAR) FOLLOWED BY A BLANK
DOES NOT REPRESENT A ONE-COMPONENT VECTOR!

WE SHALL LEARN IN MODULE 17 HOW TO MAKE IT ONE.

SO, IF 'X17' IS A SCALAR, THEN:
```
      ⍴X17
```

RESULTS IN NO PRINTED DIMENSION,
BECAUSE A SCALAR HAS NONE.

WOULD YOU LIKE ANOTHER ONE?
YES

A VECTOR WAS ASSIGNED TO THE VARIABLE "MIN",
FIND ITS LENGTH:
```
⍴MIN
2
```
RIGHT!
DISPLAY THE ELEMENTS IN MIN:

```
MIN
45 98

GOOD - HOW ABOUT THIS:

SOMETHING WAS ASSIGNED TO THE VARIABLE "X45".
WHAT'S ITS LENGTH?
ρX45

THAT'S RIGHT!

FIND OUT WHAT'S IN X45:
X45
446

A SCALAR!!!


WOULD YOU LIKE ANOTHER ONE?
NO

IF YOU WANT TO ENTER LITERAL (CHARACTER) DATA (NAMES
AND ADDRESSES, FOR EXAMPLE), YOU MAY USE ANY VALID
KEYBOARD SYMBOL, INCLUDING THE SPACE (BLANK).

YOU WILL RECALL (MODULE 4) THAT LITERAL DATA IS
ENTERED BY ENCLOSING THE CHARACTER STRING IN SINGLE
QUOTES, WHICH ARE THEN STRIPPED BY APL PRIOR TO
DISPLAY.   FOR EXAMPLE:

     A←'AREA IS'
     A
AREA IS

ONE CHARACTER ENCLOSED BY QUOTES IS A SCALAR.
ALL OTHER SEQUENCES ARE VECTORS, AND BEHAVE
AS SUCH.
SO, IF:   A←'AREA IS'
THEN:     ρA
7
I'LL ASSIGN A NEW CHARACTER SET TO  A:
     A←'THE ANSWER IS:'
     A
THE ANSWER IS:

HOW MANY ELEMENTS HAS A?
14
CORRECT!
WOULD YOU LIKE ANOTHER ONE?
NO

IF THE QUOTE SYMBOL IS TO BE PART OF THE VECTOR,
THEN IT MUST BE ENTERED TWICE.    FOR EXAMPLE:

     'DON''T DO IT'
DON'T DO IT

ENTER ANY CHARACTER VECTOR:
'ELL
 v
 HELLO THERE!'
HELLO THERE!

O.K.
```

THE SURROUNDING QUOTE SYMBOLS INDICATE THAT THE DATA THEY ENCLOSE IS TO BE USED FOR DISPLAY PURPOSES ONLY, AND THAT THEY CANNOT BE USED FOR ARITHMETIC EVALUATION. (IN A LATER MODULE WE SHALL LEARN HOW TO MANIPULATE LITERAL DATA.)

WOULD YOU LIKE TO GIVE ME ANOTHER ONE?
NO

LET'S DO SOME PROBLEMS INVOLVING VECTORS.....

IN THIS SECTION OF THE LESSON, IF YOU CANNOT ANSWER A QUESTION, JUST TYPE IN: HELP
AND YOU'LL GET THE CORRECT ANSWER PRINTED OUT.

HERE IS THE RECORD OF THREE (3) BRANDS OF CIGARETTES (A, B, AND C) YOUR FRIEND SMOKED IN THE LAST TWO (2) WEEKS:

	A	B	C
WEEK 1:	42	29	99
WEEK 2:	49	52	82

HE WOULD LIKE TO KNOW HOW MANY CIGARETTES HE SMOKED EACH WEEK REGARDLESS OF BRAND.

THIS IS HOW THE PROBLEM COULD BE SOLVED USING VECTORS:

```
      42 49 + 29 52 + 99 82
170 183
```

HE WOULD NOW LIKE TO KNOW HOW MANY CIGARETTES OF EACH BRAND HE SMOKED DURING THE TWO WEEKS.

I WOULD LIKE YOU TO ENTER AN APL EXPRESSION USING THE ABOVE NUMERIC VECTORS WHICH NOT ONLY CALCULATES THIS, BUT ALSO YIELDS THE RESULT IN BRAND SEQUENCE (THAT IS, A, THEN B, THEN C):

```
42 29 99+49 52 82
91 81 181
```

O.K.

YOUR FRIEND DECIDED TO REDUCE THE NUMBER OF CIGARETTES HE SMOKED DURING THE SECOND WEEK TO A FIXED QUANTITY OF EACH OF THE THREE BRANDS.

HERE'S HIS NEW RECORD OF CIGARETTES SMOKED:

0	A	B	C
WEEK 1:	10	79	75
WEEK 2:	13	13	13

ONCE AGAIN, HE'D LIKE TO KNOW HOW MANY CIGARETTES OF EACH BRAND HE SMOKED.

ENTER AN APL EXPRESSION WHICH WILL CALCULATE

THIS - AND SEE IF YOU CAN FIND A SHORT-CUT APPROACH.....THINK CAREFULLY:

```
10 79 75+13
23 92 88
```

VERY GOOD!

WOULD YOU LIKE TO SOLVE ANOTHER SIMILAR PROBLEM?
YES

```
            A       B       C
WEEK 1:    85      73      84
WEEK 2:    16      16      16

85 73 84+16
101 89 100
```

O.K.

A NUMERIC VECTOR (SOMETIMES CALLED A CONSTANT VECTOR) IS TREATED AS A SINGLE ENTITY BY APL.

SO, WHEN ALL THE ELEMENTS OF SUCH A VECTOR ARE IDENTICAL, IT SUFFICES TO ENTER ONLY ONE OF THEM.

THUS:

```
      85 73084+16
101 89 100
```

WOULD YOU LIKE TO SOLVE ANOTHER SIMILAR PROBLEM?
NO

WELL, OUR FRIEND DECIDED TO DROP ONE BRAND.
HERE'S HIS NEW RECORD:

```
            A       B       C
WEEK 1:    29      17      44
WEEK 2:    11      11       -
```

HOW MANY CIGARETTES OF EACH BRAND DID HE SMOKE?

```
29 17 44+11 11 0
40 28 44
```

WELL DONE!
WOULD YOU LIKE TO SOLVE ANOTHER SIMILAR PROBLEM?
NO

WHENEVER YOU WANT TO CREATE A VECTOR FROM VARIABLES, YOU CAN DO SO IN THIS MANNER:

LET'S ASSUME THAT: A←4 *AND* Y←8 *AND* Z←11

THEN, TO CREATE A VECTOR: 4 8 11 *WE WOULD ENTER:*

```
      A,Y,Z
4   8  11
```

THE COMMA CATENATES THE ELEMENTS TOGETHER, AND WE'LL LEARN MORE ABOUT IT IN LESSON 17.

GENERALLY SPEAKING, THE WAYS IN WHICH YOU CAN OPERATE ON VECTORS

ARE (AND '∘' STANDS FOR ANY FUNCTION YOU ALREADY KNOW):

1. VECTOR ← SCALAR ∘ VECTOR
2. VECTOR ← VECTOR ∘ SCALAR
3. VECTOR ← VECTOR ∘ VECTOR

AND, IN CASE 3, THE VECTORS MUST HAVE EQUAL LENGTHS.

ANDOTHAT COMPLETES LESSON 7 - YOU WILL LEARN MORE ABOUT VECTORS IN LESSONS 10 AND 11. HOWEVER, THE NEXT MODULE IS "46 LESSON8" WHICH DEALS WITH LOGICAL AND RELATIONAL PRIMITIVE FUNCTIONS.

```
      )LOAD 46 LESSON 8
INCORRECT COMMAND
      )LOAD 46 LESSON8
SAVED  18.46.37 10/06/75
```

LESSON 8

THIS MODULE PROVIDES INSTRUCTION IN THE FOLLOWING PRIMITIVE RELATIONAL AND LOGICAL FUNCTIONS:

`<  ≤  =  ≥  >  ≠  ∨  ∧  ~`

START

LESSON 8

THIS LESSON IS THE FIRST OF 2 DISCUSSING RELATIONAL AND LOGICAL PRIMITIVE FUNCTIONS, AND HERE ARE THOSE WE SHALL LEARN ABOUT IN THIS MODULE:

`<  ≤  =  ≥  >  ≠  ∨  ∧  ~`

THESE FUNCTIONS TEST THE TRUTH OF A STATEMENT, AND ALWAYS RETURN A 1 IF THE RELATIONSHIP IS TRUE, OR 0 IF FALSE.

THEY WORK LIKE THIS:

A<B	YIELDS:	1 IF A IS LESS THAN B; 0 OTHERWISE
A≤B	''	1 IF A IS LESS THAN OR EQUAL TO B; 0 OTHERWISE
A=B	''	1 IF A IS EQUAL TO B; 0 OTHERWISE
A≥B	''	1 IF A IS GREATER THAN OR EQUAL TO B; 0 OTHERWISE
A>B	''	1 IF A IS GREATER THAN B; 0 OTHERWISE
A≠B	''	1 IF A IS NOT EQUAL TO B; 0 OTHERWISE
A∨B	''	1 IF EITHER A OR B IS EQUAL TO ONE; 0 IF BOTH A AND B A
A∧B	''	1 IF BOTH A AND B ARE EQUAL TO 1; 0 IF EITHER A OR B IS
~A	''	1 IF A IS EQUAL TO 0; 0 IF A IS EQUAL TO 1

HERE ARE SOME EXAMPLES OF EXPRESSIONS USING THESE FUNCTIONS:

```
      5=6
0

      1∨0
1

      1∧0
0

      ~0
1

      ¯0.5≥34
0

      1≠2
1
```

```
      3>5
0

      1<6
1

      7≤5
0
```

THE BALANCE OF THIS LESSON CONSISTS OF EXPRESSIONS LIKE THE ABOVE EXAMPLES, AND YOU HAVE TO ENTER ANY ONE OF:

```
      1 -- IF YOU THINK THE RELATIONSHIP IS TRUE
      0 -- IF YOU THINK THE RELATIONSHIP IS FALSE
   HELP -- IF YOU DON'T KNOW
   STOP -- IF YOU'VE HAD ENOUGH
```

THE FUNCTIONS ∧ (AND) ∨ (OR) AND ~ (NOT) REQUIRE LOGICAL ARGUMENTS - THAT MEANS THEIR ARGUMENTS MAY BE ONLY 1'S OR 0'S.

IF YOU DON'T TYPE "STOP", THE LESSON WILL AUTOMATICALLY TERMINATE AFTER YOU'VE ANSWERED 50 PROBLEMS, AND, WHEN IT DOES, YOU'LL BE TOLD YOUR PERCENTAGE OF CORRECT ANSWERS.

WHENEVER YOU'RE READY TO BEGIN, TYPE: READY

```
       READY

      ¯5<4

1
```

CORRECT!

```
      ¯2>¯9

1
```

GOOD!

```
      0∨0

0
```

RIGHT!

```
      1∨1

1
```

CORRECT!

```
      1<¯1

0
```

GOOD!

```
      0∨1

1
```

RIGHT!

```
      ¯3≠10
1
CORRECT!
      ~1
0
GOOD!
      ¯8≠¯4
1
RIGHT!
      6>¯4
1
CORRECT!
      ¯6>9
0
GOOD!
      9>¯3
1
RIGHT!
      0=9
0
CORRECT!
      1≥9
0
GOOD!
      ~1
0
RIGHT!
      7≤4
0
CORRECT!
      1∨1
1
GOOD!
```

```
      ¯5=1
0
RIGHT!
      7≥¯4
1
CORRECT!
      1>¯6
1
GOOD!
      0∧0
0
RIGHT!
      1∨1
1
CORRECT!
      7≤5
0
GOOD!
      ¯8=¯1
0
RIGHT!
      ~0
1
CORRECT!
      8≠4
1
GOOD!
      ¯2>2
0
RIGHT!
      ~1
0
CORRECT!
```

```
      ¯1≤¯4
0
GOOD!
      ~0
1
RIGHT!
      3≤3
1
CORRECT!
      ¯5>¯5
0
GOOD!
      9≥9
1
RIGHT!
      0∧0
0
CORRECT!
      ¯2=6
0
GOOD!
      ¯3>9
0
RIGHT!
      1∨1
1
CORRECT!
      0>7
0
GOOD!
      ~1
0
RIGHT!
```

```
      1∨1
1
CORRECT!
      ¯4>¯7
1
GOOD!
      0∧1
0
RIGHT!
      1∧0
0
CORRECT!
      8>¯8
1
GOOD!
      1∨1
1
RIGHT!
      ~0
1
CORRECT!
      10≤¯3
0
GOOD!
      8≥7
1
RIGHT!
      ~1
0
CORRECT!
      ¯9<5
1
GOOD!
```

```
AND THAT'S THE END OF LESSON 8!

SCORE: 100 %/% (50 CORRECT IN 50 ATTEMPTS)

ATTENTION
READY[16]
      →16
THE NEXT MODULE IS "46 LESSON9", IN
WHICH THIS TOPIC WILL BE CONTINUED.
```

```
      )LOAD 46 LESSON9
SAVED  18.46.58 10/06/75
```

LESSON 9

THIS MODULE DISCUSSES EXTENDED USES
OF RELATIONAL AND LOGICAL FUNCTIONS.

START

LESSON 9

IN THIS MODULE WE SHALL EXAMINE SOME EXTENDED USES OF
THE LOGICAL AND RELATIONAL FUNCTIONS DISCUSSED IN THE
LAST LESSON.

TO BEGIN, LET'S DEFINE 2 MORE PRIMITIVE FUNCTIONS:

NAND - WRITTEN: ⍲ (~ OVERSTRUCK BY ∧)
NOR - WRITTEN: ⍱ (~ OVERSTRUCK BY ∨)

AND THEY MEAN "NOT AND" AND "NOT OR" RESPECTIVELY.

THEY WORK LIKE THIS:

A⍲B IS THE SAME AS: ~A∧B
A⍱B IS THE SAME AS: ~A∨B

AND, OF COURSE, THE ARGUMENTS MUST BE LOGICAL TYPE
DATA (1 OR 0).

THIS MODULE WILL CONSIST OF UP TO 50 PROBLEMS OF THIS TYPE:

```
      (5≥¯3)⍱4≠6
```

AND YOU HAVE TO ENTER WHAT YOU THINK APL'S EVALUATION
OF THE EXPRESSION WILL BE.

NO "HELP" IS PROVIDED IN THIS MODULE, AND YOU WILL BE
PERMITTED ONE ATTEMPT ONLY AT EACH EXPRESSION.

THE LESSON WILL TERMINATE AFTER 50 PROBLEMS, OR IF
YOU ENTER: STOP
INSTEAD OF THE ANSWER. EITHER WAY, YOU WILL RECEIVE
A "SCORE" INDICATING YOUR PERCENTAGE OF CORRECT ANSWERS.

GO.....

```
      (22=¯9)=4=20
1
```

CORRECT!

```
      (8≤8)⍲3=16
1
GOOD!
      (7≠¯1)⍲1>9
1
RIGHT!
      (15<¯6)≠0>11
0
CORRECT!
      (14=10)∨1<21
1
GOOD!
      (4=¯5)=4>23
1
RIGHT!
      (24=¯9)∨¯4≤3
1
CORRECT!
      (18<4)⍲¯1≤6
1
GOOD!
      (15≠2)=0≥10
0
RIGHT!
      (15<¯1)⍲¯3>24
1
CORRECT!
      (8≤¯7)⍲0≥7
1
GOOD!
      (12≤7)∧2<16
0
RIGHT!
```

```
      (24<¯1)⍲¯4≤10
1
CORRECT!
      (10<¯6)=4<11
0
GOOD!
      (21≥¯3)∨3<7
1
RIGHT!
      (22≤9)⍱¯2>24
1
CORRECT!
      (24≥6)⍱2>12
0
GOOD!
      (24≠5)⍱4≥16
0
RIGHT!
      (15≥¯7)∧1≤22
1
CORRECT!
      (18=5)⍲3≥22
1
GOOD!
      (18<0)∧4<19
0
RIGHT!
      (23≠9)≠2=13
1
CORRECT!
      (8≥¯5)⍲0<14
0
GOOD!
```

```
      (6≥7)∨0≠13
1
RIGHT!
      (20<¯9)∧2<25
0
CORRECT!
      (21≤4)∨0≤19
0
GOOD!
      (6>6)=¯4=18
1
RIGHT!
      (7≥9)⍲¯1<6
1
CORRECT!
      (5≤¯1)∨3≤18
0
GOOD!
      (10<7)≠5≤1
0
RIGHT!
      (6≤10)=4≤16
1
CORRECT!
      (19=6)∨2≤3
0
GOOD!
      (18≤¯5)∧¯4>17
0
RIGHT!
      (18≤2)∧1=3
0
CORRECT!
```

```
      (19>7)⍲0<4
0
GOOD!
      (11>2)≠1<8
0
RIGHT!
      (7≥0)∨¯1≥7
1
CORRECT!
      (19>4)⍱¯2≥2
0
GOOD!
      (13≥¯3)⍲0≠19
0
RIGHT!
      (9=¯6)⍲2≥17
1
CORRECT!
      (14>¯4)=4>14
0
GOOD!
      (12=7)≠5≤23
1
RIGHT!
      (15≥5)⍱¯3<21
0
CORRECT!
      (1≤6)≠2≠19
0
GOOD!
      (11=6)⍲¯3<13
1
RIGHT!
```

)9-05-

```
      (6≥1)=1>1
0
```

CORRECT!

```
      (21≤1)∨2>10
1
```

GOOD!

```
      (20>¯7)=¯1=10
0
```

RIGHT!

```
      (2≤5)=¯4<6
1
```

CORRECT!

```
      (21≤¯6)∨1≥12
0
```

GOOD!

AND THAT'S IT FOR RELATIONAL AND LOGICAL FUNCTIONS!

SCORE: 100 %/% (50 *CORRECT IN* 50 *ATTEMPTS*)

THE NEXT MODULE IS "46 *LESSON*10", *IN WHICH WE'LL DISCUSS* ***REDUCTION*** *AND* ***SCAN***.

```
      )LOAD 46 LESSON19
                      v
                      0
SAVED  18.24.47 10/06/75
```

LESSON 10

IN THIS MODULE WE SHALL LEARN ABOUT REDUCTION AND SCAN, AND CONTINUE OPERATIONS UPON VECTORS.

```
      START
```


LESSON 10

IN THIS MODULE WE SHALL LEARN ABOUT REDUCTION AND SCAN, AND CONTINUE OPERATIONS UPON VECTORS.

'REDUCTION' REDUCES THE COMPONENTS OF A NUMERIC VECTOR TO A SINGLE ELEMENT WHICH REPRESENTS THEIR SUM, PRODUCT, MAXIMUM, MINIMUM, ETC., AS THE CASE MAY BE. IT IS, IN EFFECT, A SHORT-HAND WAY OF INSERTING THE SAME PRIMITIVE FUNCTION BETWEEN ALL THE ELEMENTS OF A NUMERIC VECTOR.

'SCAN' RETAINS THE DIMENSIONS OF THE ARGUMENT, BUT APPLIES THE PRIMITIVE FUNCTION TO CONTINUING SUBSETS OF THE ARGUMENT.

THE FORMAT FOR REDUCTION IS:

    (PRIMITIVE FUNCTION) / ARGUMENT

WHILE THE SCAN FORMAT IS:

    (PRIMITIVE FUNCTION) \ ARGUMENT

AND NOW, LET'S LOOK AT SOME EXAMPLES:

```
      V←1 2 3 4 5
      +/V
15
```

WHICH APL EXECUTED AS IF IT HAD BEEN:

```
      1 + 2 + 3 + 4 + 5
```

AND:

```
      +\V
1  3  6 10 15
```

WHICH APL EXECUTED AS IF IT HAD BEEN:

```
    (1) (1+2) (1+2+3) (1+2+3+4) (1+2+3+4+5)
```

NOTICE THE RELATIONSHIP BETWEEN 'REDUCTION' AND 'SCAN' IF WE DEFINE THE EXECUTION OF THE ABOVE 'SCAN' EXAMPLE TO BE:

```
      (+/1) (+/1 2) (+/1 2 3) (+/1 2 3 4) (+/1 2 3 4 5)
```

SOME EXAMPLES OF REDUCTION AND SCAN OPERATIONS:

+/	*PLUS*	*REDUCTION*	(= *SUMMATION*)
-/	*NEGATIVE*	*REDUCTION*	(= *ALTERNATING SUM*)
+\	*PLUS*	*SCAN*	(= *SUMSCANNING*)
×\	*TIMES*	*SCAN*	(= *PRODUCT-SCANNING*)
×/	*TIMES*	*REDUCTION*	(= *PRODUCT*)
⌈/	*MAXIMUM*	*REDUCTION*	(= *LARGEST*)
∧/	*'AND'*	*REDUCTION*	

AND SO ON.

'PLUS SCAN' WILL PROVE VERY USEFUL WHEN GRAPHING A SERIES OF COORDINATES, AS WE SHALL SEE IN A LATER MODULE WHEN WE LOOK AT SCAN IN GREATER DETAIL.

IN THIS MODULE, WE'RE GOING TO SOLVE SOME REVIEW PROBLEMS ON VECTORS, AND THEN SOLVE SOME EVERYDAY PROBLEMS INVOLVING THE OPERATION OF REDUCTION OF NUMERIC VECTORS.

IF YOU GET STUCK, TYPE: HELP

HERE GOES.....

IN ONE WEEK A MAN SMOKED 86 97 59 *CIGARETTES OF* 3 *BRANDS, WHICH COST, RESPECTIVELY,* 3 5 8 *CENTS PER CIGARETTE. WHAT'S THE TOTAL COST FOR EACH BRAND?*

```
86 97 59×3 5 8
258 485 472
```
O.K.

A COCKTAIL RECIPE REQUIRES 3 4 1 2 5 *UNITS OF* 5 *LIQUORS FOR* 8 *PEOPLE HOW MANY UNITS DO YOU NEED TO MAKE ONE DRINK?*

```
3 4 1 2 5÷8
0.375 0.5 0.125 0.25 0.625
```
O.K.

YOU'VE JUST PURCHASED AN ODD LOT OF 5 *BOARDS, WHOSE LENGTHS ARE* 10 6 8 9 12 *FEET. HOWEVER, YOU STOREROOM CAN ONLY HOLD BOARDS OF* 7 *FEET OR LESS, THE LONGER ONES MUST BE TRIMMED TO THIS LENGTH. WRITE AN APL EXPRESSION WHICH CALCULATES THEIR LENGTHS AFTER TRIMMING:*

HELP
IT WOULD BE GOOD IF YOU TRIED TO ANSWER THE PROBLEM; HOWEVER, IF YOU CAN'T, TYPE "HELP" AGAIN.

HELP
```
10 6 8 9 12 ⌊ 7
```
02- *6 STUDENTS EARNED GRADES OF* 55 56 40 46 86 81 *IN A TEST, AND, UPON RE-TEST, SCORED* 69 64 49 75 97 83 *RESPECTIVELY.*

ENTER AN APL EXPRESSION WHICH SELECTS ONLY THE HIGHER SCORE FOR EACH STUDENT:

```
55 56 40 46 86 81⌈69 64 49 75 97 83
69 64 49 75 97 83
```
O.K.

YOU ARE A STORE OWNER WHO HAS A NUMBER OF ACCOUNTS WITH `2 ¯1 ¯3 3 1 0 ¯2` *DOLLAR BALANCES. YOU WOULD LIKE TO KNOW WHICH ACCOUNTS ARE OVERDRAWN.*

WRITE AN APL EXPRESSION WHICH NOT ONLY DETERMINES WHICH ACCOUNTS ARE LESS THAN ZERO, BUT ALSO RETURNS A LOGICAL VECTOR AS THE RESULT:

```
2 ¯1 ¯3 3 1 0 ¯2<0
0 1 1 0 0 0 1
```
O.K.

YOU'VE JUST PURCHASED AN ODD LOT OF 3 *BOARDS, WHOSE LENGTHS ARE* 12 6 10 *FEET. HOWEVER, YOU STOREROOM CAN ONLY HOLD BOARDS OF* 7 *FEET OR LESS, THE LONGER ONES MUST BE TRIMMED TO THIS LENGTH. WRITE AN APL EXPRESSION WHICH CALCULATES THEIR LENGTHS AFTER TRIMMING:*

```
12 6 10⌊7
7 6 7
```
O.K.

THOSE WERE SOME EXAMPLES WHICH SHOWED YOU HOW TO SOLVE PROBLEMS WITH TWO VECTORS, OR ONE VECTOR AND A SCALAR.

IT IS POSSIBLE TO SOLVE PROBLEMS WITH MORE THAN TWO ARGUMENTS IN A SIMILAR MANNER, <u>*PROVIDED*</u> *THEY ARE SCALARS OR VECTORS OF THE* <u>*SAME*</u> *LENGTH.*

RECALL OUR SMOKER:

THE CIGARETTES HE SMOKED:

```
      CIG←72 79 99 51
```
THE RESPECTIVE COSTS:

```
      COST←1 7 4 3
```
THE COST PER BRAND VECTOR:

```
      CIG×COST
72 553 396 153
```
HE NOW WANTS TO KNOW THE TOTAL COST FOR ALL THE CIGARETTES HE SMOKED, SO WE MUST ADD UP ALL THE ELEMENTS OF THIS VECTOR.

03- *LET ME ASSIGN THE COST PER BRAND VECTOR TO A VARIABLE:*

```
      TOTAL←CIG×COST
```

```
     TOTAL
72 553 396 153
```

IT IS POSSIBLE TO SUM ALL THE ELEMENTS OF "TOTAL" BY:

```
     +/TOTAL
1174
```

THE REDUCTION SYMBOL (/) MEANS THAT THE FUNCTION TO ITS LEFT IS APPLIED TO ALL THE ELEMENTS TO ITS RIGHT.

IN THE EXAMPLE, APL EXECUTED "+/TOTAL" AS IF IT WERE:

```
      72 + 553 + 396 + 153
```

HOW ABOUT THIS:

YOU HAVE A RECTANGULAR BOX WHOSE DIMENSIONS ARE 2 15 3 INCHES, AND YOU WANT TO FIND ITS VOLUME IN CUBIC INCHES.

YOU'RE GOING TO DO THIS IN TWO STEPS.

FIRST: ASSIGN THE DIMENSIONS TO A VARIABLE OF YOUR CHOICE:

```
DIM←2 15 3
```

O.K. - NOW FIND THE VOLUME (DON'T ASSIGN):

```
×/DIM
90
```

O.K.
THE SYMBOL '/' REDUCES A NUMERIC VECTOR TO A SINGLE COMPONENT REPRESENTING THEIR SUM, PRODUCT, MAXIMUM, ETC., AS THE CASE MAY BE.

IT IS CALLED <u>REDUCTION</u>.

TRY THIS:

'DUEBAL' IS THE NAME OF THE VECTOR WHICH CONTAINS THE ACCOUNT BALANCES FOR ALL THE CUSTOMERS OF YOUR STORE.

```
      DUEBAL← 5.65  7.40 12.15 22.90 19.90  5.15 29.65
```

YOU WOULD LIKE TO DETERMINE THE LARGEST AMOUNT OWED.

ENTER AN APL EXPRESSION WHICH DETERMINES THIS:

```
⌈/DUEBAL
 29.65
```

O.K.
NOW YOU WANT TO KNOW IF THERE ARE ANY NEGATIVE BALANCES - HOW DO YOU FIND THAT?

YOU MAY ASSIGN, BUT DON'T ASSIGN THE FINAL EXPRESSION:

```
DUEBAL<0
```

04-

```
0 0 0 0 0 0 0
```

O.K.
THIS IS THE VECTOR WHICH INDICATES THE NEGATIVE
BALANCES. NOW DETERMINE IF THERE ARE ANY:

```
∨/DUEBAL<0
0
```

GOOD - THE 0 SHOWS YOU THAT THERE ARE NO NEGATIVE BALANCES.

AND THE NUMBER OF NEGATIVE BALANCES CAN BE OBTAINED
BY 'PLUS' (INSTEAD OF 'OR') REDUCTION:

```
      +/DUEBAL<0
```

'ROOT' IS A VECTOR WHICH CONTAINS THE SQUARES
OF THE ROOTS OF A SET OF EQUATIONS:

```
      ROOT← 1.481 ¯1.394 2.483 0.196 4.899 0.385
```

YOU WANT TO DETERMINE WHICH OF THEM IS THE SMALLEST.

ENTER AN APL EXPRESSION WHICH DETERMINES THIS:

```
⌊/ROOT
 ¯1.394
```

GOOD!
NOW DETERMINE IF EVERY EQUATION HAS REAL ROOTS:

YOU MAY ASSIGN, BUT DON'T ASSIGN THE FINAL EXPRESSION:

```
ROOT≥0

1 0 1 1 1 1
```

O.K. - THIS IS THE VECTOR WHICH SHOWS THE REAL ROOTS.
NOW FIND OUT IF EVERY ROOT IS REAL:

```
∧/ROOT≥0
0
```

GOOD - THE 0 SHOWS YOU THAT AT LEAST ONE ROOT IS NOT REAL.

AND THAT'S IT FOR THE SECOND OF THREE MODULES
ON VECTORS. THE NEXT MODULE (46 LESSON11) COMPLETES
THE TOPIC.

```
      )LOAD 46 LESSON11
SAVED  18.25.30 10/06/75
      START
```

LESSON 11

NUMBERS, INCLUDING SCALARS AND VECTORS, CAN BE "RESHAPED" INTO OTHER DIMENSIONS BY USING THE DYADIC FORM OF: ρ

THE GENERAL FORM:

```
      MρN
```

GENERATES MANY THINGS - SO LET'S LOOK AT SOME EXAMPLES:

```
      5ρ1 2
1  2  1  2  1
```

NOTICE THAT THE RIGHT ARGUMENT, N, IS AUTOMATICALLY REPEATED (IF NECESSARY) TO FILL THE DIMENSION SPECIFIED BY "M".

```
      3ρ⍳25
1  2  3
```

NOTICE, ALSO, THAT ONLY THE FIRST "M" COMPONENTS OF "N" ARE USED (AGAIN, IF NECESSARY).

NOW, IF "M" HAPPENED TO BE A VECTOR ITSELF, WE WOULD GENERATE AN ARRAY WHOSE RANK (NUMBER OF DIMENSIONS, ONE MIGHT SAY) WOULD BE EQUAL TO THE NUMBER OF COMPONENTS IN "M".

FOR EXAMPLE:

```
      2 3ρ⍳6
  1  2  3
  4  5  6
```

AND:

```
      3 4ρ'THISBOOK'
THIS
BOOK
THIS
```

NOW WE'LL DO SOME PROBLEMS ON RESHAPING VECTORS.

USE "ρ" TO GENERATE A VECTOR OF DIMENSION 20
WHOSE COMPONENTS CONSISTS OF THE ELEMENTS OF VECTOR 45 33 8 35
⎕:

```
      20ρ45 33 8 35
)1-  CORRECT!
YOU SPECIFIED:
45 33 8 35 45 33 8 35 45 33 8 35 45 33 8 35 45 33 8 35
```

THAT WAS QUITE EASY, WASN'T IT?
YOU'LL HAVE A CHANCE TO USE ⍴
ON LARGER RANK ARRAYS IN LESSONS
17 *AND* 18.

THERE'S ANOTHER PRIMITIVE FUNCTION WHICH IS VERY USEFUL FOR GENERATING VECTORS. IN FACT, IT'S CALLED THE "INDEX GENERATOR" (ALSO "IOTA" (THE GREEK "I") BECAUSE IT'S SYMBOL IS: ⍳).
FOR EXAMPLE:

```
      ⍳N
```

GENERATES A VECTOR OF LENGTH "N" WHOSE COMPONENTS ARE:

```
      1  2  3  4  ∘  ∘  ∘  N
```

AND THE FIRST COMPONENT OF AN ***INDEX VECTOR*** *IS ALWAYS THE SAME AS THE INDEX ORIGIN* (1 *OR* 0) *WHICH WE'LL LEARN MORE ABOUT IN LESSON* 19.

HERE ARE SOME EXAMPLES OF INDEX VECTORS:

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

      ⍳3
1 2 3

      ⍳1
1
```

GENERATE AN INDEX VECTOR OF LENGTH 12
⍞:

```
      ⍳12
```

GOOD!

ARITHMETIC CAN BE PERFORMED UPON INDEX VECTORS, TOO - THEY'RE JUST LIKE OTHER VECTORS.
FOR EXAMPLE:

```
4 5 6 7 8 9 10
```

CAN BE GENERATED BY:

```
      3+⍳7
```

AND:

```
¯2 ¯1 0 1 2 3 4
```

CAN BE GENERATED BY:

```
      ¯3+⍳7
```

NOW YOU TRY A FEW.....

ENTER AN EXPRESSION (USING "⍳") WHICH GENERATES:
3 4 5 6 7
⍞:

```
      2+⍳5
```

VERY GOOD!

ENTER AN EXPRESSION (USING "⍳") WHICH GENERATES:

```
¯8 ¯7 ¯6 ¯5 ¯4 ¯3
□:
      ¯9+⍳6
```

VERY GOOD!

USE "⍳" TO GENERATE THIS SERIES OF ODD NUMBERS:

```
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29

□:
      ¯1+2×⍳15
```

OK.
HERE'S A DIFFERENT QUESTION:
FIND THE DIMENSION OF VECTOR "A":

```
□:
      ⍴A
0
```

.....SURPRISED????
IT REALLY IS A VECTOR WHOSE DIMENSION IS ZERO!
IT'S CALLED A "NULL" OR "EMPTY"
VECTOR, AND I CREATED IT BY USING: `⍳0`
WHICH IS: `⍳N` *(WHERE* `N←0`*).*

NOW THIS VECTOR (THE NULL VECTOR) WILL BE
VERY USEFUL LATER ON WHEN WE DISCUSS "BRANCHING",
SO REMEMBER IT!

WE OFTEN NEED TO SELECT A PARTICULAR COMPONENT OF A VECTOR.
THIS WE CAN DO BY SPECIFYING THE INDEX (POSITION) OF THE
DESIRED COMPONENT IN THE VECTOR. THE FORMAT FOR AN INDEX
IS THE VECTOR NAME, FOLLOWED BY THE INDEX SURROUNDED BY
SQUARE BRACKETS.

FOR EXAMPLE:

```
      V←0 11 1 4 8 6
      V[4]
4

      V[2]
11

      (⍳10)[3]
3

      (8+⍳10)[8]
16
```

NOW YOU TRY SOME:

USE INDEXING TO SELECT COMPONENT 5 *FROM VECTOR "V":*

```
□:
      V[5]
```

VERY GOOD!

USE INDEXING TO SELECT COMPONENT 8 *FROM VECTOR "V":*

```
□:
      V[8]
```

VERY GOOD!

03-

LOOKS LIKE YOU'VE GOT THE HANG OF THAT.....

```
IF YOU WISH TO SELECT MORE THAN 1 COMPONENT
THE INDEX, ITSELF, MAY BE A VECTOR.

FOR EXAMPLE:

      (V←1 2 7 9 43 18 0 8)[1 7 8]
1  0  8

      V[1 3 5]
1   7  43

      V[8 1 5]
8   1  43

YOUR TURN AGAIN:

USE INDEXING TO SELECT COMPONENTS 2 8 4 2 5 FROM VECTOR "V":
⍞:
      V[2 8 4 2 5]

RIGHT!

USE INDEXING TO SELECT COMPONENTS 4 4 8 FROM VECTOR "V":
⍞:
      V[4 4 8]

RIGHT!


NOW - USE INDEXING TO SELECT ALL THE ODD-NUMBERED
COMPONENTS OF VECTOR "V" - AND ALL I'LL TELL YOU
IS THAT ⍴V IS 75.

⍞:
      V[¯1+⍳38]
INDEX ERROR
⍞:    V[¯1+⍳38]
      ^
⍞:
      V[¯1+⍳38]
INDEX ERROR
⍞:    V[¯1+⍳38]
      ^
⍞:
      HELP

THIS ONE'S JUST LIKE ONE YOU SOLVED A FEW MINUTES AGO!
REMEMBER WHERE YOU HAD TO GENERATE A VECTOR WHOSE
COMPONENTS WERE: 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 ?
YOU CAN SOLVE THIS PROBLEM THE SAME WAY!
NOW TRY IT AGAIN.

⍞:
      V[¯1+2×⍳38]

VERY GOOD!
HERE'S THE RESULT:
13 32 13 63 25 39 3 43 95 14 10 8 26 79 35 81 66 68 26 48 61 76 96 4
AND "V" WAS:
13 20 32 63 13 66 63 81 25 48 39 21 3 91 43 15 95 42 14 89 10 17 8 3
 82 76 47 96 64 44 83 69 71 99 96 86 29 54 52 11 42 58 88 45 73 87 7

04-

NOW LET'S EXAMINE THE DYADIC USE OF IOTA (⍳).
DYADIC ⍳ IS VERY USEFUL FOR ANSWERING THESE
```

QUESTIONS:

1. *IS "N" (SCALAR OR VECTOR) A COMPONENT OF VECTOR "V"?*
2. *IF IT IS, WHICH COMPONENT IS IT?*

THE GENERAL FORM OF DYADIC IOTA:

```
V⍳N
```

YIELDS ONE OF:

1. *THE INDEX OF THE FIRST OCCURENCE OF "N" IN "V".*
2. *1+⍴V IF "N" DOES NOT OCCUR IN "V"*

AND THIS IS SO SIMPLE THAT I'M GOING TO LET YOU TRY IT YOURSELF RIGHT NOW - BUT IN A MANNER YOU HAVE NOT YET SEEN IN THIS COURSE!

THERE'S A VECTOR IN THIS WS CALLED "V", AND I WANT TO KNOW IF 7 IS A COMPONENT OF THAT VECTOR.

YOU MAY ENTER ANY EXPRESSIONS YOU CHOOSE, AND I'LL EVALUATE THEM FOR YOU. WHEN YOU THINK YOU KNOW,
ENTER: YES -- IF 7 IS IN V
OR: NO -- IF V DOESN'T CONTAIN A 7

IF YOU ENTER ANYTHING OTHER THAN "YES" OR "NO", AFTER I'VE EVALUATED IT FOR YOU, I'LL PRINT:

KEEP TRYING!

UNTIL YOU ENTER "YES" OR "NO".

ALL YOURS.....

```
[]:
      7⍳V
RANK ERROR
[]:   7⍳V
      ^
[]:
      V⍳7
7
KEEP TRYING!
[]:
      YES
0
KEEP TRYING!
[]:
      ⍴V
6
KEEP TRYING!
[]:
      NO
1
```

RIGHT!
7 IS NOT IN V, BECAUSE V←⍳6 AND: V⍳7 YIELDS: 7 (1+⍴V).

NOW LET'S TRY ANOTHER PROBLEM:
SAME RULES AS THE LAST ONE, EXCEPT INSTEAD OF A "YES" OR "NO", I'D LIKE YOU TO ENTER AN APL EXPRESSION:

WHAT'S THE INDEX OF 20 IN V? (A NEW V.)

```
[]:
```

```
      V←⍳20
26

CORRECT.
BECAUSE V WAS:
18 5 23 22 23 14 4 12 25 6 12 8 13 23 11 12 21 10 6 25 4 16 16 1 1

DYADIC ⍳ CAN BE USED ON LITERAL VECTORS.
ALSO, THE RIGHT ARGUMENT MAY ITSELF BE A VECTOR,
IN WHICH CASE THE RESULT WILL BE A VECTOR OF INDICES.
THE LEFT ARGUMENT, HOWEVER, MUST ALWAYS BE A VECTOR.


LET'S EXAMINE 2 MORE PRIMITIVE FUNCTIONS:

      ↑ -- "TAKE"     AND     ↓ -- "DROP"

   N↑V  TAKES (SELECTS)  THE FIRST N COMPONENTS OF V
   N↓V  DROPS (DISCARDS) THE FIRST N COMPONENTS OF V

IF "N" IS A POSITIVE INTEGER.  IF "N" IS NEGATIVE, CHANGE
"FIRST" TO READ "LAST".

HERE ARE SOME EXAMPLES:

      3↑V←⍳10
  1  2  3

      3↓V
  4  5  6  7  8  9 10

      ¯3↑V
  8  9 10

      ¯3↓V
  1  2  3  4  5  6  7

NOW IT'S YOUR TURN (AND NO "HELP"!).
HERE'S ANOTHER "V":
15 7 9 14 14 5 17 15 17 2 2 16 13 5 5 2 8 20 19 8

TAKE THE FIRST 10 COMPONENTS OF V
□:
      10↑V

DROP THE LAST 15 COMPONENTS OF V
□:
      ¯15↓V

DROP THE FIRST 19 COMPONENTS OF V
□:
      19↓V

VERY GOOD!
YOU SHOULD NOW BE ABLE TO ANSWER THIS LAST QUESTION:

EVALUATE:

     ∧/(¯10↑V)=10↓V←⍳20

□:
      1

EXCELLENT!
```

AND THAT'S ALL WE SHALL SAY ABOUT VECTORS.

THE NEXT MODULE IS "46 LESSON12M" WHICH, DESPITE ITS "M" SUFFIX, REALLY WON'T BE VERY DIFFICULT, BECAUSE TYPING: HELP IF YOU'VE FORGOTTEN A FORMULA WILL GET YOU AS MUCH ASSISTANCE AS YOU'LL NEED.

HOWEVER, IF YOU'RE STILL CHICKEN, YOU MAY PROCEED DIRECTLY TO "46 LESSON13".

```
      )LOAD 46 LESSO12M
SAVED  13.36.46 02/17/75
      START

COPYRIGHT 1971, IBM CORPORATION.

YOU ARE NOW FAMILIAR WITH VECTORS, PRIMITIVE FUNCTIONS, AND SPECIFICA
LET'S TAKE A LOOK AT SOME APPLICATIONS OF THESE THINGS COMBINED.....


HERE'S A PROBLEM:

WHAT IS APL'S EVALUATION OF:

     3-4-(6-(2-7))-8

BE CAREFUL!
2

OK. LET'S CONTINUE.....


THERE IS A VECTOR, R, IN THIS WS.
DISPLAY IT:

R
15 15 20 18 5

GOOD!

NOW I'M GOING TO CHANGE THE VALUE OF "R" (WE'LL USE THE NEW VALUE LAT


THERE IS A FUNCTION  "○"  (UPPER CASE "O")
WHICH WILL BE DISCUSSED IN GREATER DETAIL IN LESSON 22M.
FOR NOW, JUST REMEMBER THAT THE EXPRESSION:

     ○X

MEANS:      ○ × X  (WHERE ○ IS 3.141592654)

YOU WILL RECALL THAT I SAID THAT I'D CHANGED THE VALUE OF "R" - WELL,
CHANGED IT TO BE THE RADIUS OF A CIRCLE.

NOW, SPECIFY "P" AS THE CIRCUMFERENCE OF A CIRCLE OF RADIUS "R":

□:
      P←○2×R

GOOD!
YOUR "P" IS: 43.98229715


NOW, LET'S GO A LITTLE FARTHER.....

01-  SPECIFY "A○" TO BE THE AREA OF A CIRCLE OF RADIUS "R".

NOTE: FROM NOW ON, IF A QUESTION CATCHES YOU OFF GUARD, TYPE:  HELP
```

```
□:
      A0←○R×R

VERY GOOD!
YOUR "A0" IS: 153.93804


NOW SPECIFY "A1" TO BE THE AREA
OF THE SQUARE INSCRIBED BY THE CIRCLE.

□:
      A1←4×T
           v

          v
          R×R

VERY GOOD.  YOU'RE COMING ALONG THERE.


HERE'S ONE THAT'S A LITTLE HARDER:
SPECIFY "A2" AS THE AREA OF THE SQUARE
CIRCUMSCRIBED BY THE CIRCLE.
□:
      A2←2×R×R
GOOOOOOOOOOOOD!


NOW FOR A CHANGE OF PACE!

WHAT IS THE DIFFERENCE BETWEEN THE AREAS OF THE TWO SQUARES?
(REMEMBER - THEIR INDIVIDUAL AREAS HAVE ALREADY BEEN SPECIFIED!)
□:
      A1-A2
OK.     BUT THAT PROBLEM WAS TOO EASY.


HERE'S ESSENTIALLY THE SAME PROBLEM, BUT MADE MORE DIFFICULT:
"P", "A0", "A1", AND "A2" HAVE BEEN ERASED,
AND I'VE CHANGED THE VALUE OF "R" (THE RADIUS).

NOW - IN ONE LINE (ONE APL EXPRESSION) FIND THE
DIFFERENCE BETWEEN THE INSCRIBED AND CIRCUMSCRIBED SQUARES
□:
      2×R×R
VERY GOOD.
YOU'RE GOOD AT CIRCLES AND SQUARES.


HERE'S A DIFFERENT PROBLEM - NOT TOO DIFFICULT:
IMAGINE A RIGHT TRIANGLE, WITH SIDES "A", "B", AND "C",
WHERE SIDE "C" IS THE HYPOTENUSE.
I'VE ALREADY SPECIFIED "A" AND "B" -
NOW YOU (IN ONE LINE) FIND "C".
□:
      C←((A×A)+B×B)*.5
EXCELLENT!  MR. PYTHAGORAS COMMENDS YOU.

NOW SPECIFY "AREA" TO BE THE AREA OF THE TRIANGLE.
□:
      AREA←.5×A×B
```

$\sqrt{a^2+b^2}$.

RIGHT!

NOW LET'S TRY A HARD ONE!
IN ONE LINE, SPECIFY "AC" AS THE AREA OF
THE CIRCLE WHICH CIRCUMSCRIBES
OUR TRIANGLE - REMEMBER, ONE LINE!

```
⍞:
      AC←○(.5×((A×A)+B×B)*.
          ∨
          ○((A×A+B×B)÷4
SYNTAX ERROR
⍞:    AC←○((A×A+B×B)÷4
                    ∧
⍞:
      AC←○((A×A)+B×B)÷4
```

GOOD!

ENOUGH GEOMETRY - LET'S TRY ALGEBRA.
IF JOHN HAD "J" DOLLARS, AND HE GAVE "F" TO
FRANK AND "M" TO MARY,
HOW MANY DOES HE HAVE LEFT?
("J", "F", AND "M" ARE ALREADY SPECIFIED.)

```
⍞:
     (J-F)-M
     ∨
     J-F+M
```

CORRECT!
THIS PROBLEM SEEMS SIMPLE, BUT, AS YOU KNOW,
THERE IS A RIGHT-TO-LEFT SCAN TRAP.

HERE'S THE FINAL PROBLEM:
WHAT IS THE RESULT (I.E., APL'S EVALUATION) OF:

```
     2+2 2+2
```

```
6 6
```

WELL DONE!

YOU HAVE JUST BEEN THROUGH A SET OF A DOZEN PROBLEMS DESIGNED
TO FAMILIARIZE YOU WITH WRITING APL EXPRESSIONS.

IN ALL OF THEM, YOU HAD TO REMEMBER:

EXECUTION IS FROM RIGHT TO LEFT

AND YOU PROBABLY WON'T FORGET THAT!

THE NEXT MODULE IS "46 LESSON13", IN WHICH WE SHALL START
TO DISCUSS DEFINED FUNCTIONS.

```
      )LOAD 46 LESSON13
SAVED  18.26.51 10/06/75
      START
```


APL HAS MANY BUILT-IN FUNCTIONS, SOME QUITE POWERFUL. BUT IT IS UNLIKELY THAT THESE WILL MEET EVERYONE'S NEEDS, SO APL ALSO ALLOWS US TO DEFINE OUR OWN FUNCTIONS. THESE DEFINED FUNCTIONS MAY BE AS COMPLEX AS WE CARE TO MAKE THEM, SO THAT FUNCTION DEFINITION IN APL IS ANALOGOUS TO PROGRAM WRITING IN ANOTHER LANGUAGE.

NOW, IF WE ARE TO BE ABLE TO DEFINE FUNCTIONS, WE MUST HAVE SOME WAY OF LETTING THE SYSTEM KNOW THAT WE DON'T WANT OUR DEFINITION STATEMENTS EXECUTED EACH TIME WE HIT THE "RETURN" KEY - IN OTHER WORDS, WE MUST BE ABLE TO DIFFERENTIATE BETWEEN DEFINITION AND EXECUTION MODE.

SO, WE ENTER DEFINITION MODE BY TYPING: ∇ (UPPER CASE "G") AND FOLLOWING IT WITH THE DESIRED FUNCTION NAME. FOR EXAMPLE:

```
     ∇FUNCTION1
```

TO WHICH APL WILL RESPOND:

```
[1]
```

TO INDICATE THAT THE SYSTEM IS READY FOR YOUR FIRST LINE OF THE FUNCTION "FUNCTION1".

YOU CAN NOW ENTER A LINE (SAY: X←3⍴0) AND HIT "RETURN", WHEREUPON THE SYSTEM WILL RESPOND WITH: [2], THUS:

```
[1]   X←3⍴0
[2]
```

AND YOU PROCEED THUS UNTIL YOU HAVE ENTERED ALL YOU WANTED TO, AFTER WHICH YOU ENTER ANOTHER "∇" (CALLED 'DEL') TO LEAVE DEFINITION MODE AND RETURN TO EXECUTION MODE. SOMETIMES, WE SAY THAT DEFINITION IS EITHER "OPEN" OR "CLOSED", SO BE PREPARED TO SEE THESE TERMS.

TO PUT IT ALL TOGETHER:

```
      ∇FUNCTION1
[1]   X←3⍴0
[2]   Y←+/(X[1]←2×A),(X[2]←A÷2),X[3]←A*2
[3]   Y
[4]   ∇
```

THIS IS ONLY AN EXAMPLE - THE FUNCTION IS REALLY USELESS! BUT IT ILLUSTRATES THE METHOD.

NOW - I WANT YOU TO START THE DEFINITION OF A FUNCTION CALLED "WOW":

```
∇WOW
```

CORRECT!

I'VE ALREADY DEFINED A FUNCTION IN THIS WS CALLED: HYP
AND YOU'RE GOING TO DISPLAY IT - HERE'S WHAT YOU DO:

```
      ∇HYP[□]∇
```

WHICH APL INTERPRETS AS:

```
∇           ENTER DEFINITION MODE
∇HYP        WITH RESPECT TO FUNCTION "HYP"
∇HYP[□]     DISPLAY THE ENTIRE FUNCTION
∇HYP[□]∇    RETURN TO EXECUTION MODE
```

NOW YOU DO IT - AND DON'T FORGET TO ENTER THE CLOSING "∇"!!!

WHEN YOU'VE DONE THAT, TYPE: CONTINUE

```
      ∇HYP[□]∇
    ∇ C←A HYP B
[1]   C←((A*2)+B*2)*0.5
    ∇
      CONTINUE
```

AS YOU SAW, FUNCTION "HYP" CALCULATES THE HYPOTENUSE OF A RIGHT TRIANGLE ACCORDING TO THEOREM OF A FAMOUS GREEK. HOWEVER - NOTICE THE HEADER LINE (LINE [0] - WHERE ITS NAME IS DEFINED):

```
      C←A HYP B
```

THE NAME "HYP" IS SIMILAR TO THE PRIMITIVE FUNCTION IN A DYAD. "A" AND "B" ARE CALLED DUMMY VARIABLES BECAUSE THEY ONLY HAVE VALUES UPON EXECUTION, AND BEHAVE LIKE THE ARGUMENTS OF A DYAD.

HOWEVER, WE HAVE ALSO SPECIFIED "C" TO BE THE EXPLICIT RESULT OF THE EXECUTION OF "HYP" (WHICH MEANS WE MUST GIVE IT A VALUE SOMEWHERE IN THE FUNCTION!).

LET ME DEMONSTRATE:

```
      3 HYP 4
5
```

JUST LIKE A PRIMITIVE DYAD, ISN'T IT?

THIS FORM OF DEFINED FUNCTION IS, THEREFORE, CALLED:

DYADIC, WITH EXPLICIT RESULT

I'VE ANOTHER FUNCTION IN THIS WS, CALLED: HYPN
WHICH PERFORMS THE SAME CALCULATION, IS ALSO DYADIC,
BUT HAS NO EXPLICIT RESULT.

DISPLAY IT NOW, AND COMPARE IT TO THE DISPLAY OF "HYP".

WHEN YOU'RE READY, TYPE: READY

```
      ∇HYPN[□]∇
    ∇ A HYPN B
[1]   C←((A*2)+B*2)*0.5
    ∇
      READY
```

SINCE "HYPN" HAS NO EXPLICIT RESULT, WE SHALL HAVE TO REQUEST A DISPLAY OF VARIABLE "C" IN ORDER TO LEARN THE OUTCOME OF ITS EXECUTION.

NOW, YOU EXECUTE BOTH FUNCTIONS (BY ENTERING: ARGUMENT HYP ARGUMENT) AND PROVE FOR YOURSELF THAT WHAT I JUST SAID IS, IN FACT THE CASE.

WHEN YOU'RE READY FOR MORE, TYPE: MORE

```
      5 HYP 12
13
      C←5 HYPN 12
VALUE ERROR
      C←5 HYPN 12
      ^
      5 HYPN 12
      C
13
      MORE
```

WE CAN ALSO DEFINE A MONADIC FUNCTION, WITH OR WITHOUT EXPLICIT RESULTS.

HOW MANY ARGUMENTS DO YOU THINK SUCH A FUNCTION WOULD HAVE?

```
□:
      1
```

RIGHT!

I'M NOW GOING TO WRITE A FUNCTION WHICH IS VERY SIMPLE, BUT DOES ILLUSTRATE THE POINT:

```
      ∇A←SQUARE B
[1]   A←B*2
[2]   ∇
```

OR I COULD HAVE PUT THE CLOSING "∇" AFTER THE LAST CHARACTER IN LINE[

```
      ∇A←SQUARE B
[1]   A←B*2∇
```

EITHER WAY IS ACCEPTABLE.

IT IT ALSO POSSIBLE TO DEFINE FUNCTIONS WITH NO ARGUMENTS, AGAIN WIT OR WITHOUT EXPLICIT RESULTS. THIS TYPE OF FUNCTION IS CALLED NILAD

HERE'S AN EXAMPLE:

```
      ∇CALCULATE
[1]   'THIS IS A SAMPLE FUNCTION ONLY.'
[2]   Y←(X←3?4+⍵B)*0.5∇
```

AND NOW IT'S YOUR TURN....

13- HERE'S A LIST OF FUNCTIONS I'D LIKE YOU TO WRITE:

FUNCTION NAME CALCULATION TO BE PERFORMED

TRIAREA	*AREA OF A TRIANGLE*
RECTAREA	*AREA OF A RECTANGLE*
RECTPERIM	*PERIMETER OF A RECTANGLE*
DIST	*DISTANCE TRAVELLED (I.E., VELOCITY × TIME)*
SQRTFN	*SQUARE ROOT OF A POSITIVE INTEGER*
CIRCUM	*CIRCUMFERENCE OF A CIRCLE*
CIRCAREA	*AREA OF A CIRCLE*
SPHEREVOL	*VOLUME OF A SPHERE*

AND HERE ARE THE RULES:

1. *ALL FUNCTIONS MUST HAVE EXPLICIT RESULTS - IF THEY DON'T, I SHAN'T KNOW TO WHICH VARIABLE YOU'VE ASSIGNED THE RESULT, AND I WON'T BE ABLE TO CHECK IT FOR YOU.*

2. *IF YOU'VE FORGOTTEN ANY OF THESE SIMPLE FORMULAE, TYPE:*

   CANTDO 'FUNCTIONNAME' (E.G., CANTDO 'SPHEREVOL')

   AND I'LL DISPLAY AN EXAMPLE OF HOW THAT FUNCTION COULD BE DONE.

3. *AFTER EACH FUNCTION HAS BEEN WRITTEN, AND YOU'VE TESTED IT, LET ME CHECK IT BY TYPING:*

   CHECK 'FUNCTIONNAME' (E.G., CHECK 'TRIAREA')

IMPORTANT! THE ARGUMENTS OF "CHECK" AND "CANTDO" ARE LITERALS - SO BE SURE TO ENCLOSE THEM IN SINGLE QUOTES!

AND, IN CASE YOU FORGOT, AND ONE OF YOUR FUNCTIONS DOES NOT HAVE AN EXPLICIT RESULT, YOU CAN RECOVER (FOR THIS LESSON ONLY) FROM THE RESULTING PROGRAM INTERRUPTION AS FOLLOWS:

1. *TYPE IN A RIGHT ARROW (→), THEN HIT "RETURN".*
2. *ERASE YOUR FUNCTION - THE ONE WHICH CAUSED THE INTERRUPTION BECAUSE IT HAD NO EXPLICIT RESULT.*
3. *RE-DEFINE THE FUNCTION - THIS TIME WITH AN EXPLICIT RESULT.*
4. *LET ME CHECK IT AGAIN.*

BEFORE I LEAVE YOU TO THE RIGORS OF FUNCTION DEFINITION, LET ME EASE YOUR BURDEN BY STATING THAT I HAVE ALREADY DEFINED A VARIABLE "PI" TO BE 22÷7.

AND, WHEN YOU'VE FINISHED THEM ALL, AND I'VE CHECKED THEM, TYPE: FINISHED

```
      ∇C←A TRIAREA B
[1]   C←.5×A×B
[2]   ∇
      CHECK 'TRIAREA'
```

YOUR "TRIAREA" FUNCTION CHECKS OUT OK.

```
      ∇C←A RECTAREA B
[1]   C←A×B∇
      CHECK 'RECTAREA'
```

YOUR "RECTAREA" FUNCTION CHECKS OUT OK.

```
      ∇C←∇
DEFN ERROR
      ∇C←∇
          ^
```

```
      CANTDO 'RECTPERIN
                        v
                        M'
```

ONE WAY OF WRITING FUNCTION "RECTPERIM" WOULD BE:

```
     ∇P←L RECTPERIM W
[1]   P←2×L+W
     ∇
      ∇C←A RECTPERIM B
[1]   'HET GAAT HIER DUS OM DE OMTRE!'
                                      v
                                      EK!'
[2]   C←2×A+B∇
      1 RECTPERIM 1
HET GAAT HIER DUS OM DE OMTREK!
4
      CHECK 'RECTPERIM'
HET GAAT HIER DUS OM DE OMTREK!
```

YOUR "RECTPERIM" FUNCTION CHECKS OUT OK.

```
      ∇C V DIST T
DEFN ERROR
  0   ∇C V DIST T
                ^
      ∇C←V DIST T
[1]   C←V×T∇
      CHECK 'DIST'
```

YOUR "DIST" FUNCTION CHECKS OUT OK.

```
      ∇SQRT←SQRTFN A
[1]   SQRT←A*.5∇
      CHECH 'SQRTFN'
SYNTAX ERROR
      CHECH'SQRTFN'
      ^
      CHECK 'SQRTFN'
```

YOUR "SQRTFN" FUNCTION CHECKS OUT OK.

```
      ∇C←CIRCY
             v
             UM R
[1]   C←2×PI×R
[2]   ∇
      CHECK 'CIRCUM'
```

YOUR "CIRCUM" FUNCTION CHECKS OUT OK.

```
      ∇C←CIRCAREA R
[1]   C←P×R×R∇
      CHECK 'CIRCAREA'
VALUE ERROR
CIRCAREA[1]  C←P×R×R
              ^
      )ERASE CIRCAREA
SI DAMAGE
      ∇C←CIRCAREA R
[1]   C←PI×R×R∇
      CHECK 'CIRCAREA'
```

YOUR "CIRCAREA" FUNCTION CHECKS OUT OK.

```
      ∇V←SPHEREVOL R
[1]   V←÷3×4×R*3∇
      CHECK 'SPHEREVOL'

YOUR "SPHEREVOL" FUNCTION YIELDS AN INCORRECT RESULT.
ERASE, THEN RE-DEFINE IT.

      )ERASE SPHEREVOL
      ∇V←SPE
           ∨
           HEREVOL R
[1]   V←÷3×4×PI×R*3
[2]   ∇
      CHECK ⍳'
            ∨
            'SPHEREVOL'

YOUR "SPHEREVOL" FUNCTION YIELDS AN INCORRECT RESULT.
ERASE, THEN RE-DEFINE IT.

      )ERASE SPHEREVOL
      ∇V←SPHEREVOL R
[1]   V←(4×PI×R*3)÷3
[2]   ∇
      CHECK 'SPHEREVOL'

YOUR "SPHEREVOL" FUNCTION CHECKS OUT OK.

      FINISHED

THIS CONCLUDES THE FIRST OF 2 MODULES ON DEFINED FUNCTIONS;
THE NEXT ONE WILL BE "46 LESSON14" WHICH CONTINUES THE TOPIC.
```

```
      )LOAD 46 LESSON14
SAVED  18.27.46 10/06/75
      START
```

LESSON 14

WHEN YOU HAVE A DEFINED FUNCTION, IT CAN BE MADE AVAILABLE FOR GENERAL USE IF YOU PUT IT INTO WHAT IS CALLED "CONVERSATIONAL" FORM.

REMEMBER THE "HYP" FUNCTION FROM LESSON 13? WELL, I'VE ONE IN THIS MODULE, BUT I'VE CALLED IT "HYPC", AND IT'S A CONVERSATIONAL VERSION OF "HYP".

NOW -- DISPLAY: HYPC; EXAMINE IT; AND, WHEN YOU'RE READY TO GO ON, TYPE: CONTINUE

```
      ∇HYPC[[]]∇
    ∇ HYPC
[1]   ''
[2]   'THIS FUNCTION CALCULATES THE HYPOTENEUSE OF A RIGHT'
[3]   'TRIANGLE, GIVEN THE LENGTHS OF THE SIDES.'
[4]   ''
[5]   'ENTER THE LENGTH OF ONE OF THE SIDES:'
[6]   X←⍞
[7]   ''
[8]   'ENTER THE LENGTH OF THE OTHER SIDE (SAME UNIT OF MEASURE):'
[9]   Y←⍞
[10]  Z←X HYP Y
[11]  ''
[12]  'THE LENGTH OF THE HYPOTENEUSE'
[13]  '(IN THE SAME UNITS) IS:'
[14]  ''
[15]  Z
    ∇
      HYPC

THIS FUNCTION CALCULATES THE HYPOTENEUSE OF A RIGHT
TRIANGLE, GIVEN THE LENGTHS OF THE SIDES.

ENTER THE LENGTH OF ONE OF THE SIDES:
⍞:
      1

ENTER THE LENGTH OF THE OTHER SIDE (SAME UNIT OF MEASURE):
⍞:
      1

THE LENGTH OF THE HYPOTENEUSE
(IN THE SAME UNITS) IS:

1.414213562
      CONTINUE
```

I'VE GOT 2 MORE CONVERSATIONAL VERSIONS OF THIS FUNCTION IN THIS WS.
01- THEY ARE: HYPC1 AND HYPC2

I WOULD NOW LIKE YOU TO:

```
1. EXECUTE ALL 3 CONVERSATIONAL FUNCTIONS (YOU DO THIS BY TYPING
   IN THE NAME OF THE FUNCTION YOU WANT EXECUTED).
2. OBSERVE THE DIFFERENCES IN THE INPUT AND OUTPUT FORMATS.
3. DISPLAY THEM, AND COMPARE THEIR STRUCTURES.

WHEN YOU'VE DONE THAT, AND ARE READY TO PROCEED WITH INSTRUCTION,
TYPE:  READY

      HYPC1
THE HYPOTENUSE OF A RIGHT TRIANGLE
IS SPECIFIED BY THE SQUARE ROOT OF
THE SUM OF THE SQUARES OF THE LEGS.

GIVE ME THE LENGTH OF A LEG OF A RIGHT TRIANGLE.
□:
      1
GIVE ME THE LENGTH OF THE OTHER LEG (IN THE SAME UNITS).
□:
      1
THE LENGTH OF THE HYPOTENUSE (IN THE SAME UNITS) IS 1.414213562
      HYPC2

FUNCTION TO CALCULATE THE HYPOTENEUSE OF A
RIGHT TRIANGLE, GIVEN THE LENGTHS OF THE SIDES.

ENTER THE LENGTH OF SIDE 1:
□:
      1

THE SQUARE OF THE SIDE WHOSE LENGTH IS 1 IS 1.

NOW SIDE 2 (SAME UNITS):
□:
      1

SIDE 2 IS 1 UNITS LONG.
ITS SQUARE IS: 1.

THE HYPOTENEUSE IS, THEREFORE, 1.414213562.

      ∇HYPC1[□]∇
    ∇ HYPC1
[1]   'THE HYPOTENUSE OF A RIGHT TRIANGLE'
[2]   'IS SPECIFIED BY THE SQUARE ROOT OF'
[3]   'THE SUM OF THE SQUARES OF THE LEGS.'
[4]   ''
[5]   'GIVE ME THE LENGTH OF A LEG OF A RIGHT TRIANGLE.'
[6]   X←□
[7]   'GIVE ME THE LENGTH OF THE OTHER LEG (IN THE SAME UNITS).'
[8]   Y←□
[9]   Z←X HYP Y
[10]  'THE LENGTH OF THE HYPOTENUSE (IN THE SAME UNITS) IS ';Z
    ∇
      ∇HYPC2[□]∇
    ∇ HYPC2
[1]   ''
[2]   'FUNCTION TO CALCULATE THE HYPOTENEUSE OF A'
[3]   'RIGHT TRIANGLE, GIVEN THE LENGTHS OF THE SIDES.'
[4]   ''
[5]   'ENTER THE LENGTH OF SIDE 1:'
[6]   X←□
[7]   ''
[8]   'THE SQUARE OF THE SIDE WHOSE LENGTH IS ';X;' IS ';X*2;'.'
[9]   ''
```

```
      1
        0]  'NOW SIDE 2 (SAME UNITS):'
      [11]  Y←⍞
      [12]  ''
      [13]  'SIDE 2 IS ';Y;' UNITS LONG.'
      [14]  'ITS SQUARE IS: ';Y*2;'.'
      [15]  ''
      [16]  'THE HYPOTENEUSE IS, THEREFORE, ';Z←X HYP Y;'.'
      [17]  ''
          ∇
            READY
```

IN LESSON 13, YOU WROTE THESE DEFINED FUNCTIONS:

TRIAREA	RECTAREA	RECTPERIM	SPHEREVOL
DIST	SQRTFN	CIRCUM	CIRCAREA

AND TO SAVE YOU SOME BOTHER, I'VE ALREADY COPIED THEM INTO THIS WS.

HOWEVER, EVERY ONE OF THEM HAS AN EXPLICIT RESULT, AND SHOULD BE USED LIKE THIS:

```
[?]   'DISTANCE TRAVELLED = ';V DIST T
```

WHEN YOU CONVERT THEM TO CONVERSATIONAL FORM - WHICH YOU'RE JUST ABOUT TO DO!

USE FUNCTION NAMES DIFFERENT FROM THE ABOVE LIST, AND TRY YOUR CONVERSATIONAL FUNCTIONS OUT TO SEE IF THEY WORK AS YOU INTEND.

WHEN YOU HAVE WRITTEN THEM ALL, AND ARE SATISFIED WITH THEIR EXECUTION, TYPE: RESUME

```
      ∇DIST1
[1]   'THIS FUNCTION CALCULATES
                              ∨

SYNTAX ERROR
[1]   'THIS FUNCTION CALCULA
      ∧
[2]   [1]
[1]   'DEZE FUNCTIE BEREKENT DE AFGELEGDE WEG,'
[2]   'GEGEVEN DE SNELHEID EN
                           ∨
                           VAN EEN VOERTUIG EN'
[3]   'DE TIJD GEC
                 ∨
                 DURENDE WELKE HET VOERTUIG ZICH'
[4]   'VERPLAATST.'
[5]   'TYP EERST EEN WAARDE VOOR DE SNELHEID IN (M/SEC)'
[6]   V←⍞
[7]   'GEEF NU EEN WAARDE VOOR DE TIJD (SEC)'
[8]   T←⍞
[9]   'DE AFGELEGDE WEG IS NU IN METERS:';V DIST T
[10]  ∇
      DIST1
DEZE FUNCTIE BEREKENT DE AFGELEGDE WEG,
GEGEVEN DE SNELHEID VAN EEN VOERTUIG EN
DE TIJD GEDURENDE WELKE HET VOERTUIG ZICH
VERPLAATST.
TYP EERST EEN WAARDE VOOR DE SNELHEID IN (M/SEC)
⍞:
      10
GEEF NU EEN WAARDE VOOR DE TIJD (SEC)
```

```
□:
DE AFGELEGDE WEG IS NU IN METERS:30
      RESUME
```

YOU NOW HAVE A FEEL FOR THE BASICS OF WRITING CONVERSATIONAL FUNCTIONS WHICH USE THE "QUAD" SYMBOL (□) FOR EVALUATED INPUT.

YOU HAVE SEEN HOW AN EMPTY VECTOR ('') ALONE ON A LINE CAN BE USED TO SPACE PRINTED OUTPUT.

YOU HAVE SEEN THAT A LITERAL, ALONE, ON A LINE RESULTS IN ITS DISPLAY DURING EXECUTION.

YOU HAVE ALSO SEEN THAT A SEMI-COLON CAN BE USED TO SEPARATE LITERAL AND NUMERIC OUTPUT DATA TO BE PRINTED ON THE SAME LINE.

NOW LET'S LOOK AT HANDLING LITERAL INPUT.
THIS IS DONE IN A SIMILAR MANNER TO EVALUATED INPUT, ONLY YOU USE THE "QUAD-QUOTE" (⍞ - □ OVERSTRUCK BY ') INSTEAD OF "□".

YOU USE THIS TYPE OF INPUT FOR (USUALLY) PLAIN TEXT DATA.

AND HERE'S ANOTHER PROBLEM WHICH FREQUENTLY CROPS UP:

I'M WRITING THE (SAY) 27TH FUNCTION IN A WS, AND I NEED A NAME FOR A VARIABLE - WHICH NAMES HAVE I ALREADY USED?

YOU WOULD BE QUITE CORRECT TO ASSUME THAT ONCE A VARIABLE HAS BEEN SPECIFIED DURING EXECUTION, THEN ALL FUNCTIONS IN THAT WS HAVE ACCESS TO IT. SUCH A VARIABLE IS CALLED A GLOBAL VARIABLE.

BUT (SINCE I BROUGHT THE SUBJECT UP!) THERE JUST HAS TO BE A WAY TO RESTRICT A VARIABLE'S VALUE IN ANY GIVEN FUNCTION TO THAT FUNCTION ALONE!

AND, OF COURSE, THERE IS! AND - IT'S THIS SIMPLE:

IN THE FUNCTION HEADER, MERELY FOLLOW THE FUNCTION NAME (AND ARGUMENTS, IF ANY) WITH THE DESIRED VARIABLE NAMES, SEPARATED BY SEMI-COLONS, LIKE THIS:

```
     ∇HYPD;X;Y;Z
```

WHICH HAS THE EFFECT OF MAKING "X", "Y", AND "Z" LOCAL TO FUNCTION "HYPD" - THAT IS, X, Y, AND Z MAY BE USED ELSEWHERE IN THE WS WITHOUT BEING CHANGED DURING EXECUTION OF HYPD (IT'S NOT QUITE THAT SIMPLE, BUT WE'LL STICK WITH THAT DEFINITION FOR A WHILE).

SO THAT, IN OTHER WORDS, A LOCAL VARIABLE HAS NO MEANING OUTSIDE THE FUNCTION TO WHICH IT IS LOCAL.

FOR EXAMPLE: IF YOU EXECUTED "HYPD" AND ASKED FOR THE VALUES OF X, Y AFTERWARDS, YOU'D GET A "VALUE ERROR" MESSAGE IN EACH CASE. HOWEVER YOU WERE TO EXECUTE "HYPC" AND ASK FOR THEIR VALUES, YOU'D GET THEM.

LET'S TRY IT OUT AND SEE!

FIRST, SINCE YOU'VE ALREADY EXECUTED "HYPC", YOU MUST ERASE THE VARIABLES SPECIFIED DURING ITS EXECUTION. YOU DO THIS BY ENTERING:

```
     )ERASE X Y Z
```

NEXT, EXECUTE "HYPD", THEN ASK FOR THE DISPLAY OF X, Y, AND Z.

AFTER THAT, EXECUTE "HYPC" ONCE AGAIN, AND ASK FOR THE DISPLAY OF THE

AND, WHEN YOU'RE THROUGH WITH ALL THAT, TYPE: MORE

```
      X Y Z
SYNTAX ERROR
      X Y Z
      ^
      X;Y;Z;
111.414213562
      X,Y,Z
1 1 1.414213562
      X;' ',
           v
           ;Y;' ';Z
1 1 1.414213562
      )ERASE X Y Z
      HYPD
```

THIS FUNCTION CALCULATES THE HYPOTENEUSE OF A RIGHT
TRIANGLE, GIVEN THE LENGTHS OF THE SIDES.

ENTER THE LENGTH OF ONE OF THE SIDES:

```
□:
      5
```

ENTER THE LENGTH OF THE OTHER SIDE (SAME UNIT OF MEASURE):

```
□:
      12
```

THE LENGTH OF THE HYPOTENEUSE
(IN THE SAME UNITS) IS:

```
13
      X,Y,Z
VALUE ERROR
      X,Y,Z
        ^
      MORE
```

SOMETIMES IT IS USEFUL TO HAVE COMMENTS ON A LINE IN A FUNCTION
TO DESCRIBE WHAT THE FUNCTION DOES, YET WHICH ARE NOT EXECUTED.

THE SYMBOL ⍝ (UPPER CASE "C", BACKSPACE, UPPER CASE "J") WILL
CAUSE APL TO TREAT EVERYTHING TO ITS RIGHT AS A COMMENT.

TO SEE WHAT I MEAN, DISPLAY FUNCTION "HYPE", AND, WHEN YOU'RE
FINISHED, TYPE: ENOUGH

```
      ∇HYPE[□]_
               v
               ∇
    ∇ HYPE;X;Y
[1]   ⍝ THIS FUNCTION CALCULATES THE HYPOTENUSE OF A
[2]   ⍝ RIGHT TRIANGLE GIVEN THE LENGTHS OF ITS SIDES.
[3]    'HYPOTENUSE = ';(X←□,0⍴□←'ENTER SIDE 2:')HYP Y←□,0⍴□←'ENTER SID
    ∇
      HYPE
ENTER SIDE 1:
□:
      5
ENTER SIDE 2:
□:
```

```
      12
HYPOTENUSE = 13
      ENOUG
      v
      )LIB
COMPTER
DTBL2
MSDT
SCHNABEL
SEQ
TRACEY
      )VARS
AREA1   AREA2   B       CONTINUE        D       DESCRIBE        ENOUG
RESUME  START   T
      )WIDTH 60
WAS 120
      )VARS
AREA1   AREA2   B       CONTINUE        D       DESCRIBE
ENOUGH  FINISHED        H       MORE    PI      READY
RESUME  START0  T       V
      )FNS
CIRCAREA        CIRCUM  DIST    DISTV   DIST1   HYP     HYPC
HYPC1   HYPC2   HYPD    HYPE    RAREA   RECTAREA
RECTPERIM       SPHEREVOL       SQRTFN  TAREA   TRIAREA
      ENOUGH
```

YOU WILL HAVE NOTICED THE USE OF THE QUAD (⎕) TO DISPLAY ON THE SAME LINE THAT ANOTHER "⎕" IS USED FOR EVALUATED INPUT - IT WORKS, TOO!

EXECUTE FUNCTION "HYPE" (AND OBSERVE THAT LINES [1] AND [2] DO NOT PRINT) THEN, TYPE: FINISHED

```
      HYPE
ENTER SIDE 1:
⎕:
      5
ENTER SIDE 2:
⎕:
      12
HYPOTENUSE = 13
      FINISHED
```

YOU ARE WHAT YOU JUST TYPED IN!

THIS IS THE END OF LESSON 14 - WE'LL BE DISCUSSING HOW TO "BRANCH" AROUND IN A DEFINED FUNCTION IN THE NEXT MODULE, "46 LESSON15".

```
      )LOAD 46 LESSON15
SAVED  18.28.42 10/06/75
      )WIDTH 70
WAS 120
      START
```

0 LESSON 15

THE BRANCH OPERATOR (→) IS USED WITH A RIGHT ARGUMENT TO CONTROL THE FLOW OF EXECUTION OF A FUNCTION.

IN ITS SIMPLEST FORM, IT IS AN UNCONDITIONAL BRANCH TO THE LINE REPRESENTED BY ITS RIGHT ARGUMENT.

FOR EXAMPLE:

```
      →4

      →2×2
```

BOTH MEAN: EXECUTE LINE [4] NEXT.

A BRANCH TO LINE [0] OR TO LINE [X] WHERE THE VALUE OF X EXCEEDS THE NUMBER OF THE LAST LINE OF THE FUNCTION WILL TERMINATE THE EXECUTION OF THAT FUNCTION.

LET'S SEE IF YOU'VE GOT THAT.....

I'M GOING TO GIVE YOU SOME PROBLEMS. I WANT YOU TO GET CORRECT ANSWERS TO THREE PROBLEMS BEFORE YOU GO ON. THE PROBLEMS ALL DEAL WITH THE FOLLOWING FUNCTION:

```
    ∇ SAMPLE1
[1]   'THIS IS A SAMPLE FUNCTION.'
[2]   →__________
[3]   X←4
[4]   X←X+Y
[5]   X←X*3
[6]   X←X,X
[7]   →2
    ∇
```

NOTICE LINE [2] OF THE SAMPLE FUNCTION. IN EACH QUESTION, I WANT YOU TO 'FILL IN THE MISSING QUANTITY'.

CAUSE A BRANCH TO THE LINE WHICH HAS ..... X←X,X

```
□:
      6
```

CAUSE A BRANCH TO THE LINE WHICH HAS ..... X←4

```
□:
      3
```

YOU DID A
LL RIGHTOON THOSE. NOW, TRY THIS ONE:

PROVIDE A QUANTITY THAT WILL CAUSE AN EXIT FROM THE FUNCTION.
□:
8
GOOD. REMEMBER THIS. IN GENERAL, A BRANCH TO A LINE NUMBER WHICH IS NOT IN THE FUNCTION OR A BRANCH TO 'LINE [0]' CAUSES AN EXIT FROM THE FUNCTION.

ANOTHER USEFUL FACILITY TAKES ADVANTAGE OF AN EMPTY VECTOR (THE NULL VECTOR), WHICH CAN BE REPRESENTED VARIOUS WAYS, INCLUDING:

```
      ⍳0
   0⍴(ANYTHING)
U/V (WHERE ∧/~U IS 1)
```

THE FOLLOWING RULES INVOLVING ARITHMETIC ON THE NULL VECTOR ARE HANDY TO REMEMBER:

```
N+⍳0 IS ⍳0
N×⍳0 IS ⍳0
    WHERE N IS ANY NUMBER
```

THIS SPECIAL VECTOR IS USEFUL BECAUSE →⍳0 IS EQUIVALENT TO 'DON'T BRANCH' OR 'FALL THROUGH'. THE RESULT IS THAT THE NEXT LINE IN SEQUENCE IS EXECUTED. FOR EXAMPLE, ANY OF THE FOLLOWING EXAMPLE LINES MEAN BRANCH TO LINE [6] IF N=2, AND 'FALL THROUGH' TO LINE [3] OTHERWISE:

```
[2]   →6×⍳N=2
[2]   →(N=2)⍴6
[2]   →(N=2)/6
```

THE EXPRESSION "×⍳" CAN BE READ AS "IF".

FROM NOW ON, IF YOU GET STUCK, TYPE: HELP

EXAMINE THIS FUNCTION:

```
    ∇ SAMPLE3 I;A
[1]   →__________
[2]   A←I*0.5
[3]   →__________
[4]   C←A-B
[5]   →__________
[6]   C←A×B
    ∇
```

"I" IS PROVIDED UPON EXECUTION, AND WE SEE THAT WE MUST EXIT FROM THE FUNCTION IF I<0 BECAUSE WE CALCULATE A SQUARE ROOT ON LINE [2].

ENTER AN EXPRESSION WHICH EXITS IF I<0 AND "FALLS THROUGH" OTHERWISE.

□:
0×⍳I<0

CORRECT!

NOW, AFTER WE'VE COMPUTED "A" ON LINE [2], WE WANT TO COMPARE IT WITH "B" ON LINE [3].
(ASSUME THAT "B" IS A GLOBAL SCALAR.) IF A<B → [6] AND IF A≥B

→ [4] (FALL THROUGH).
ENTER AN EXPRESSION FOR LINE [3] WHICH WILL DO THIS.

□:

```
      A3
       v
       3
      v
    (A<Bρ
        v
        )ρ6
```

GREAT!

NOW, ON LINE [5] WE WANT AN UNCONDITIONAL EXIT. ENTER AN APPROPRIATE EXPRESSION:

□:

```
      8
```

GOOD FOR YOU!

COMPRESSION

ONE OPERATOR WE SHALL FIND VERY USEFUL IN BRANCHING IS / (CALLED "COMPRESS").

IT WORKS LIKE THIS:

```
    (0 0 1 0)/4 9 2 0
```

AND YIELDS: ,2 (A ONE-COMPONENT VECTOR)

WHAT HAPPENS IS THAT AN ELEMENT IN THE RIGHT-HAND ARGUMENT IS RETAINED IF ITS
CORRESPONDING ELEMENT IN THE LEFT ARGUMENT IS 1 , AND DISCARDED IF ITS
CORRESPONDING ELEMENT IS 0 .

SUPPOSE WE WANTED TO BRANCH ON THE VALUE OF "J" AND WE KNOW THAT "J" CAN BE
ONLY 1 OR 0 , THEN WE TAKE THIS EXPRESSION:

```
      (J=1 0)/LINEA,LINEB
```

AND IF WE MAKE THAT THE RIGHT ARGUMENT OF THE BRANCH OPERATOR:

```
    →(J=1 0)/LINEA,LINEB
```

THEN, IF (1=(J=1)) LINEA WILL BE EXECUTED NEXT, AND IF (1=(J=0)) LINEB WOULD BE THE NEXT ONE TO BE EXECUTED, A N D IF (0=((J=1)v(J=0))) NO BRANCH WOULD BE TAKEN - THE FLOW OF EXECUTION WOULD "FALL THROUGH" AND THE NEXT SEQUENTIAL LINE WOULD BE EXECUTED!

LET'S TRY SOME PROBLEMS.....

EVALUATE: 1/5

□:

```
      5
```

GOOD!
TRY THIS -

EVALUATE: (0 0 0)/29 74 19

(THINK CAREFULLY!)

⍞:
```
      ⍳0
```

EXCELLENT!

LET'S CONTINUE.

USING A VECTOR OF LINE NUMBERS IS OFTEN HANDY WHEN YOU DESIRE TO BRANCH TO ONE OF MANY LINES. THE PARTICULAR LINE MAY BE SELECTED USING EITHER INDEXING OR COMPRESSION AS IN:

```
      0        → 10 14 23 8[J]
```

OR

```
      →(J=1 2 3 4)/10 14 23 8
```

IMAGINE A FUNCTION WHERE IT IS DESIRED TO BRANCH TO ONE OF A NUMBER OF LINES DEPENDING UPON THE VALUE OF "I" AS INDICATED BELOW. YOU MAY ASSUME THAT OTHER VALUES FOR "I" WILL NEVER OCCUR.

FOR I =	0	1	2	3	4
BRANCH TO:	8	6	13	8	6

THE FIRST FEW LINES OF THE FUNCTION LOOK LIKE:

```
    ∇ SAMPLE4 I
[1]   →__________
[2]   Z←A+B+7
[3]     ∘
        ∘
```

NOW, PROVIDE THE INFORMATION THAT IS MISSING FROM LINE [1].
TRY TO DO IT BY INDEXING.
⍞:
```
      8 6 13 8 6[I-1]
                 ∨
                 +1]
```

GOOD!

NOW, TRY IT FOR THE FOLLOWING TABLE:

FOR I =	2	0	4	8	6	7
BRANCH TO:	8		6	13	8	6

DO THIS ONE BY COMPRESSION.
⍞:
```
      (I=2 4 8 6 7)/8 6 13 8 6
```

GOOD!

OCCASIONALLY YOU'LL WANT TO 'JUMP' A CERTAIN NUMBER OF LINES

AHEAD OR BEHIND THE LINE CONTAINING A '→'. ONE OF A SPECIAL CLASS OF FUNCTIONS CALLED 'IBEAMS' IS HANDY HERE. ⌶26 ALWAYS YIELDS THE LINE NUMBER OF THE LINE BEING EXECUTED. (NOTE: '⌶' IS FORMED BY '⊤' AND '⊥' OVERSTRUCK.) (ANOTHER NOTE: ⌶26+2 IS EQUIVALENT TO ⌶28, WHILE 2+⌶26 ADDS 2 TO THE CURRENT LINE NUMBER).

HERE'S A SAMPLE FUNCTION:

```
    ∇ SAMPLE5 I;X;Z
          ∘
          ∘
          ∘
[?]   →__________
[?+1] →0×ρρ,Z←FN1 X
[?+2] →0×ρρ,Z←FN2 X
[?+3] →0×ρρ,Z←FN3 X
          ∘
          ∘
          ∘
    ∇
```

INSERT THE INFORMATION MISSING FROM LINE [?] SUCH THAT CONTROL JUMPS TO LINE [?+2] IF I>0. (I DON'T CARE WHAT HAPPENS IF I≤0.)

```
[]:
      2+⌶26×⍳I>0
```

NOW CAUSE A JUMP TO LINE [?+I] DEPENDING ON THE VALUE FOR I.

```
[]:
      I+⌶26
```

GOOD.
NOW TRY THIS ONE:
CAUSE A JUMP TWO LINES AHEAD IF I≥0 AND THREE LINES BEHIND IF I<0.

```
[]:
      (0 1=I≥0)/¯3 2
```

VERY GOOD!
THAT'S IT FOR JUMPS.

ONE OF THE INCONVENIENCES OF USING LINE NUMBERS FOR BRANCHING IS THAT YOU'LL HAVE TO CHANGE MANY STATEMENTS IF YOU MODIFY (EDIT) THE FUNCTION BY REMOVING OR INSERTING A LINE (OR LINES).

SO, APL BORROWED AN IDEA FROM OTHER LANGUAGES, AND ALLOWS YOU TO USE

LABELS

INSTEAD OF ACTUAL LINE NUMBERS.

THEY WORK LIKE THIS:

```
        ∇SAMPLEX I;A;B;C
  [1]   →(I=⍳4)/L3,LINEFIVE,S001B3,AHA2
          ∘
          ∘
          ∘
  [?] L3:(WHATEVER IS ON LINE [?])
          ∘
          ∘
  [
```

?+N] AHA2:(WHATEVER IS ON LINE [?+N])

AND SO ON.

NOTICE THAT THE 'LABELS' REPLACE THE NUMERIC VALUES OF THE LINES; ALSO THAT
THE LABEL IS ASSIGNED TO A LINE BY STARTING THE LINE WITH THE LABEL, THEN
FOLLOWING THE LABEL WITH ':'. THIS IS THE ONLY PERMISSIBLE USE FOR THE COLON
(OTHER THAN IN A LITERAL) IN APL.

THAT'S QUITE EASY TO UNDERSTAND, SO NOW IT'S YOUR TURN.....

HERE IS A SAMPLE FUNCTION:

```
     ∇ Z←SAMPLE6 Z
[1]    →((A>0),A<0)/6 4
[2]    Z←0
[3]    →7
[4]    Z←2○A
[5]    →7
[6]    Z←1○A
[7]    Z←Z+○2
     ∇
```

IF WE WANT TO INCORPORATE LABELLING COMPLETELY IN PREFERENCE
TO INDICATING BRANCH POINTS BY THE LINE NUMBERS THEMSELVES, ON
WHICH LINES MUST THE 'LABELS' THEMSELVES BE PLACED?

```
L:   0
       4 6 7
```
RIGHT. LABELS MUST BE PUT ON LINES 4, 6, AND 7.

ON WHICH LINES MUST CHANGES BE MADE (REPLACING NUMBERS WITH LABEL NAMES)?

```
L:
       1 ,3 ,5
```
GOOD. NOW, HOW WILL YOU LABEL LINE [4]?

```
A:
```
HOW ABOUT LINE [6]?

```
B:
```
AND, LINE [7]?

```
C:
```

NOW, WE HAVE TO CHANGE SOME OTHER LINES.

WHAT GOES ON THE NEW LINE [3].

```
→C
```
GOOD. NOW, HOW ABOUT THE NEW LINE [5]?

```
→C
```
NOW, WHAT GOES TO THE RIGHT OF THE '/' ON LINE [1]?

```
B,A
```
VERY GOOD!

THE RIGHT-HAND ARGUMENT OF "→" CAN BE A COMPUTED NUMBER, WHICH WILL PROVE VERY USEFUL.

LOOK AT THIS EXAMPLE:

```
    ∇ SAMPLE2 I;BP
[1]   X←?73
[2]   Y←2○A
[3]   →__________
[4]   ⎕←Y
[5]   →0
[6]   Y←Y+X*B
[7]   →3
[8]   Y←Y+X*C
[9]   →3
    ∇
```

"I" IS PROVIDED ON EXECUTION, AND WILL ONLY HAVE THE SCALAR VALUES OF 0 OR 1.
ENTER THE EXPRESSION WHICH WILL CAUSE A BRANCH TO LINE [6] IF I=0 OR TO LINE [8] IF I=1.
(I'VE ALREADY SPECIFIED "I" AS: I←0 1 SO I CAN CHECK YOUR ANSWER.)

```
⎕:
      (6 8)[1+I=1]
```

GOOD.

NOW LET'S MAKE IT HARDER.
"I" CAN BE ANY POSITIVE INTEGER. ENTER AN EXPRESSION WHICH BRANCHES TO LINE [6] IF "I" IS ODD, OR TO LINE [8] IF "I" IS EVEN.
(I SPECIFIED I←8 9 TO CHECK YOUR ANSWER.)

```
⎕:
      (6 8)[1+(I÷2)=⌈I÷2]
```

FINE!

ONE LAST POINT ON THIS TOPIC.
GIVEN THE EXPRESSION:

```
      →5 2 5 7
```

A BRANCH TO LINE [5] WILL ALWAYS RESULT.
THIS IS BECAUSE THE BRANCH FUNCTION OPERATING UPON AN UNINDEXED VECTOR ALWAYS CAUSES A BRANCH TO THE LINE REPRESENTED BY THE FIRST COMPONENT OF THAT VECTOR.

WITH ALL YOU KNOW NOW, YOU SHOULD BE FAIRLY PROFICIENT AT DIFFERENT TYPES OF BRANCHING. I'D LIKE YOU TO TRY TWO COMMONLY ENCOUNTERED PROBLEMS.

I WANT YOU TO WRITE A FUNCTION CALLED 'CONTROL' WHICH SETS UP A LOOP AND EXECUTES THAT LOOP EXACTLY 14 TIMES. INSIDE THAT LOOP, CALL A FUNCTION 'CHECKER'. I'VE ALREADY WRITTEN 'CHECKER' AND WILL USE IT TO SEE IF YOU HAVE DONE YOUR PART PROPERLY. 'CHECKER' HAS NO EXPLICIT INPUTS OR OUTPUTS.

TYPE 'READY' WHEN YOU HAVE COMPLETED THE FUNCTION 'CONTROL' AND YOU ARE READY TO HAVE IT CHECKED.

```
      ∇CONTROL
[1]   A←0
[2]   0X⍳A=14
[3]   [2]
7-    [2]   0×I
            ∨
            ⍳A=14
```

```
[3]     [2]
[2]     →0×ιA=14
[3]     CHECKER
[4]     →2
[5]     ∇
        ∇CHECKER[4]
[4]     A←A+1
[5]     →2
[6]     ∇

        ∇CHECKE
        ∨
        ∇CONTROL[□]∇
    ∇   CONTROL
[1]     A←0
[2]     →0×ιA=14
[3]     CHECKER
[4]     →2
    ∇
        )ERASE CONTROL
        ∇CONTROL
[1]     A←0
[2]     →0×ιA=14
[3]     CHECKER
[4]     A←A+1
[5]     →2
[6]     ∇
        ∇CONTROL[□]∇
    ∇   CONTROL
[1]     A←0
[2]     →0×ιA=14
[3]     CHECKER
[4]     A←A+1
[5]     →2
    ∇
        READY
NOW I'LL CHECK IT.  HERE GOES .....

CORRECT
```

HERE'S THE SECOND PROBLEM.....

A FROG IS STRANDED AT THE BOTTOM OF A WELL. HE DECIDES TO HOP HIS WAY OUT. WITH HIS FIRST HOP, HE HOPS HALF WAY TO THE TOP. BECAUSE HE HAS IRON-POOR BLOOD, EACH SUCCEEDING HOP IS ONLY HALF AS HIGH AS THE ONE IMMEDIATELY PRECEDING.

WRITE A FUNCTION CALLED 'FROG' WHICH SIMULATES THE HOPPING FROG. EXIT FROM THE FUNCTION WHEN THE HEIGHT OF A HOP IS LESS THAN .001 TIMES THE HEIGHT OF THE WELL. PASS A VECTOR, V, EXPLICITLY OUT OF THE FUNCTION. (THAT IS, THE FUNCTION HEADER SHOULD LOOK LIKE: ∇ *V←FROG)*

V SHOULD BE CONSTRUCTED AS FOLLOWS:

```
V[1]←TOTAL NUMBER OF HOPS
V[2]←TOTAL FRACTION OF WELL TRAVERSED
V[3]←LENGTH OF LAST HOP
```

INCIDENTALLY, YOU SHOULD BE ABLE TO WRITE 'FROG' IN 2 LINES!!!

WHEN YOU'VE COMPLETED THE "FROG" FUNCTION, TYPE: DONE
AND I'LL CHECK IT OUT FOR YOU.

```
OFF YOU GO.....

      ∇V←FROG
[1]   V[1]←10
[2]   [1]
[1]   3ρV
[2]   V[1]←10
[3]   [2]
[2]   V[1]←11
[3]   V[2]←1+
            v
            +
            v
            -.5*10
[4]   V[3]←,
          v
          ∇
      )ERASE FROG
      ∇V←FROG
[1]   3ρV
      v
      V←3ρ0
[2]   V[1]←11
[3]   V[3]←.5*10
[4]   V[2]←1+
            v
            -V[]
               v
               3]
[5]   ∇
      ∇FROG[[]]∇
    ∇ V←FROG
[1]   V←3ρ0
[2]   V[1]←11
[3]   V[3]←0.5*10
[4]   V[2]←1-V[3]
    ∇
      DONE
NOW I'LL CHECK YOUR 'FROG' FUNCTION.
HERE GOES.....

YOUR FROG TOOK TOO MANY JUMPS.
FIX YOUR FUNCTION, AND I'LL CHECK IT AGAIN.

TYPE "DONE" WHEN YOU'RE READY TO HAVE IT CHECKED.

(IF YOU'RE REALLY STUCK, TYPE:  HELPME )

      )ERASE FROG
      ∇V←FROG
[1]   V←3ρ0
[2]   V[1]←10
[3]   V[3]←.5*9
[4]   [3]
[3]   V[3]←.5*10
[4]   V[2]←V[3]
      v
      V[2]←1-V[3]
[5]   ∇
      DONE
NOW I'LL CHECK YOUR 'FROG' FUNCTION.
HERE GOES.....
```



```
IT'S GOOD!
```

AS A MATTER OF INTEREST, THE "FROG" PROBLEM (WHICH IS, ITSELF, A VARIATION ON THE OLD "BOUNCING BALL" PROBLEM) CAN BE SOLVED IN 1 LINE, WITHOUT BRANCHING:

```
      ∇FROG[□]∇
    ∇ V←FROG;I;J
[1]   V←I,(1-J),J←÷2*I←⌈2⊛1000
    ∇
```

BUT IT WOULDN'T HAVE PROVIDED AN EXERCISE IN BRANCHING!

AND THAT'S ALL WE SHALL SAY ABOUT BRANCHING IN THIS MODULE - WE SHALL EXAMINE SOME MORE ADVANCED TECHNIQUES IN LESSON 21.

PROCEED NOW TO "46 LESSON16" IN WHICH WE'LL LEARN HOW TO EDIT A DEFINED FUNCTION.

```
      "load 46 lesson16
saved 018.30.22 10/06/75
      )WIDTH 70
WAS 120
      START

COPYRIGHT 1971, IBM CORPORATION.
```

F U N C T I O N E D I T I N G

```
ENTER THE OPERATOR WHICH EITHER OPENS OR CLOSES FUNCTION DEFINITION:

∇

GOOD!

YOU WILL RECALL THAT THE QUAD (⎕) IS USED TO DISPLAY THE LINES IN A FU
NCTION.
FOR EXAMPLE:

      ∇SAMPLE[⎕]∇
    ∇ SAMPLE
[1]   CR,'THIS IS A SAMPLE FUNCTION.',CR
[2]   →(((A×B÷C),(A+B+C),HELP←2⍕8)=Y←1↑⎕,0ρ⎕←'ENTER THE VALUE FOR "Y":
')/S1,S2,S3
[3]   →2,ρ⎕←'THAT''S NOT RIGHT!',CR,'TRY AGAIN, OR ASK FOR HELP:'
[4]  S3:CR,'THE CORRECT ANSWER IS: ';A×B÷C;'.'
[5]   →2,ρ⎕←'NOW ENTER IT CORRECTLY:'
[6]  S2:CR,'NOT QUITE!',CR,'YOU ENTERED: ';Y;' AND I WANTED: ';A×B÷C;'
.'
[7]   →2,ρ⎕←'TRY IT AGAIN:'
[8]  S1:CR,'CORRECT!'
      ∇

A FEW FACTS ABOUT THIS SAMPLE FUNCTION - FIRST, IT IS NOT
SUPPOSED TO HAVE ANY PURPOSE OTHER THAN AN EXAMPLE; SECOND,
THE VARIABLE "CR" IS A CARRIER-RETURN/LINE-FEED CHARACTER
(VERY USEFUL - YOU CAN COPY IT IF YOU NEED IT).
LAST, IT ILLUSTRATES SOME CODING TECHNIQUES YOU SHOULD EXAMINE.

HOWEVER, BACK TO FUNCTION EDITING!
NOTICE THAT THE EXPRESSION:

      ∇(FUNCTION NAME)[⎕]∇

RESULTS IN A DISPLAY OF THE ENTIRE FUNCTION, AND THE SYSTEM
REMAINS IN EXECUTION MODE.

BUT, HAD WE ENTERED:

      ∇SAMPLE[⎕]                    (NO CLOSING "∇")

THE RESULT WOULD HAVE BEEN:

    ∇ SAMPLE
[1]   CR,'THIS IS A SAMPLE FUNCTION.'
```

```
[2]              ∘
                 ∘
                 ∘
   0             ∘
[7]              ∘
[8]  S1:CR,'CORRECT!'
     ∇
[9]
```

WITH THE SYSTEM NOW IN DEFINITION MODE, AWAITING YOUR EDITING INSTRUCTIONS.

IF WE HAD MERELY ENTERED:

```
      ∇SAMPLE
```

THE SYSTEM WOULD RESPOND WITH:

```
[9]
```

AND THIS IS QUITE CONSISTENT - YOU'VE OPENED DEFINITION, BUT DID NOT ASK FOR A DISPLAY.

ONE LAST ITEM BEFORE YOU TRY SOME YOURSELF - IF WE HAD ENTERED:

```
      ∇SAMPLE[5]
```

THE SYSTEM WOULD HAVE RESPONDED WITH:

```
[5]
```

AND WAIT FOR YOUR INSTRUCTIONS.
IN THIS EXAMPLE, IT BECOMES EASY TO EDIT LINE [5].
THE LINE NUMBER CAN BE OVERRIDDEN EASILY, TOO.
FOR EXAMPLE:

```
      ∇SAMPLE
[9]  [8]S1:CR,'VERY GOOD!'
[9]   ∇
```

AND NOW IT'S YOUR TURN.....

SUPPOSE YOU HAD A FUNCTION NAMED 'F1' IN THIS WORKSPACE AND THAT ITS LAST LINE WAS LINE [6]. WHAT WOULD YOU TYPE TO ADD A NEW LINE TO THE END OF 'F1'?
(NOTE: FUNCTION DEFINITION IS CLOSED.)

```
∇F1
```

THEN, WHAT LINE NUMBER WILL THE SYSTEM RETURN WITH <u>AFTER</u> YOU HAVE INSERTED THAT NEW LINE?

```
[]:
      8
```

GOOD!

NOW, ASSUMING THAT THAT'S ALL THE EDITING YOU WANT TO DO, WHAT DO YOU TYPE NEXT?

```
∇
```

GOOD!
NOW WHAT'S THE NUMBER OF THE LAST LINE IN THE FUNCTION 'F1'?

```
⎕:
RIGHT!
SUPPOSE NOW THAT YOU WANTED TO GO BACK AND REPLACE LINE [2] COMPLETELY
.
HOW WOULD YOU DO IT?
∇F1[2]
GOOD.  NOW WRITE IN A NEW LINE -- ANYTHING AT ALL WILL DO.
'ALS JE ME ER NOG EEN KEER ZONDER REDEN UITGOOIT WORDT IK BOOS!'
NOW CLOSE FUNCTION DEFINITION.
∇
FINE.

VERY OFTEN YOU WILL WANT TO INSERT A NEW LINE BETWEEN TWO EXISTING LIN
ES.
ASSUMING FUNCTION DEFINITION IS NOW CLOSED, TYPE A COMMAND THAT WILL A
LLOW
YOU TO PUT A NEW LINE BETWEEN EXISTING LINES [2] AND [3] OF FUNCTION '
F1'.

∇F1[2.5]

NOW, WHAT WOULD APL'S RESPONSE BE AFTER YOU HAD ENTERED A LINE BETWEEN
LINES [2] AND [3], AND HAD HIT THE RETURN KEY?

[2.6]
GOOOOOOOOOOOD.

THERE ARE THREE SIMILAR COMMANDS WHICH CAN BE USED TO DISPLAY FUNCTION
S
OR PARTS OF FUNCTIONS --

     [⎕]     DISPLAYS THE WHOLE FUNCTION
     [N⎕]    DISPLAYS LINE [N] ONLY
     [⎕N]    DISPLAYS LINE [N] AND ALL SUCCEEDING LINES.

LET'S TRY A FEW EXERCISES USING THESE EDITING COMMANDS.

DISPLAY ALL OF FUNCTION 'F1'.
(ASSUME DEFINITION IS CLOSED NOW, BUT LEAVE IT OPEN.)
∇F1[⎕]
GOOD.

NOW, LIST THE HEADER STATEMENT ONLY OF FUNCTION 'F1'.
(THE "HEADER" IS LINE [0])
∇F1[0⎕]
ERROR:  INVALID COMMAND STRING (CHARACTERS AFTER '∇')
YOU DON'T NEED THE '∇' OR THE FUNCTION NAME -- YOU ENTERED DEFINITION
MODE WITH YOUR LAST INPUT!  THE CORRECT EXPRESSION IS:   [0⎕]  TRY IT
AGAIN.......
[0⎕]
VERY GOOD.

NOW, ASK FOR A DISPLAY OF LINE [4] AND ALL SUCCEEDING LINES OF FUNCTIO
N 'F1'.
[⎕4]
GOOD.

NOW CLOSE FUNCTION DEFINITION.
∇

OKAY, YOU SEEM TO HAVE THE HANG OF THAT.


TO EDIT WITHIN A LINE ITSELF, WE HAVE A POWERFUL SET OF TOOLS.
```

FIRST, WE CAN POSITION THE TYPE HEAD ANYWHERE (AT LEAST, FROM 1 TO 130
)
ALONG THE LINE TO BE EDITED.

FOR EXAMPLE, IF WE WANTED TO CHANGE THE 34TH CHARACTER IN LINE [7]
OF FUNCTION 'SAMPLE', WE'D ENTER:

```
      ∇SAMPLE[7⎕34]
```

WHICH HAS THIS EFFECT -

1. LINE [7] IS DISPLAYED.
2. THE CARRIAGE FEEDS 1 LINE.
3. THE TYPE HEAD IS POSITIONED AT THE 34TH CHARACTER POSITION
   FROM THE LEFT MARGIN.

WE CAN NOW ENTER ONE (OR MORE) OF THESE EDITING TOOLS:

/ UNDER A CHARACTER WILL DELETE IT
1 UNDER A CHARACTER WILL ADD 1 BLANK (SPACE) IMMEDIATELY TO
THE LEFT OF THE UNDERSTRUCK CHARACTER. (SIMILARLY FOR DIGITS 2 TO 9
)
A (OR B THROUGH I) WILL WILL INSERT 5×LETTER VALUE SPACES.
(A=5, B=10, C=15, ETC.)

FOLLOWING THIS, THE LINE WILL BE DISPLAYED, WITH THE DESIRED CHARACTER
S
DELETED, AND SPACES INSERTED AS WAS REQUESTED. YOU MAY NOW FILL IN TH
E
BLANKS (SPACES) WITH ANYTHING YOU WISH!

LET ME GIVE YOU AN EXAMPLE.

LINE [7] OF FUNCTION 'F1' LOOKS LIKE THIS:

```
[7]   →2,X←X+2
```

AND WE WANT IT TO READ: [7] →2,X←X+1
SOOOOOO.....

```
      ∇F1[7⎕14]
[7]   →2,X←X+2
      ⍝        /1
```

AND THE RESPONSE WILL BE:

```
[7]   →2,X←X+
```

WITH THE TYPE HEAD POSITIONED ALONGSIDE THE "+" IN LINE [7].

JUST TYPE IN THE "1", AND CLOSE DEFINITION, LIKE THIS:

```
[7]   →2,X←X+1∇
```

ANOTHER EXAMPLE:

```
      ∇F1[25⎕25]
[25]  'THIS IS AN EXAMPLE OF EDITING LITERAL DATA.'
```

THE TYPE HEAD WOULD COME TO REST UNDER THE "E" OF "EXAMPLE".
BUT THAT'S NOT WHAT WE WANT - WE WANT TO CHANGE THE WORDS
"AN EXAMPLE" TO READ "A SAMPLE".

AIN NO PROBL
EM!

SIMPLY BACKSPACE TO THE DESIRED CHARACTER - IT'S THAT EASY - AND THEN TYPE THE DESIRED EDIT CHARACTERS, LIKE THIS:

```
[25]  'THIS IS AN EXAMPLE OF EDITING LITERAL DATA.'
                / //1
[25]  'THIS IS A  AMPLE OF EDITING LITERAL DATA.'   (SYSTEM RESPONSE)
```

AFTER WHICH THE TYPE HEAD COMES TO REST IN THE EDITED LINE, AND AT THE FIRST INSERT POSITION (WHICH, IN THIS CASE, NEATLY ALLOWS US TO TYPE IN AN "S", HIT "RETURN", THEN CLOSE DEFINITION).

AND HERE'S A PROBLEM FOR YOU:

EDIT WITHIN LINE [6] OF 'F1'. HAVE THE SYSTEM POSITION THE CARRIER SO MEWHERE BETWEEN THE 46TH AND 52ND CHARACTERS.

```
∇F1[6□49]
```

VERY GOOD!

SUPPOSE YOU WANTED TO DELETE A LINE FROM A FUNCTION?
NO PROBLEM AT ALL! JUST HIT "ATTN" THEN "RETURN"
ALONGSIDE THE LINE NUMBER TO BE DELETED.

FOR EXAMPLE:

TO DELETE LINE [4] FROM FUNCTION 'F1':

```
      ∇F1[4]
[4]
       ^
[5]    ∇
```

AND THE LINES IN THE FUNCTION WOULD BE RE-NUMBERED CONSECUTIVELY FOLLOWING THE CLOSING "∇".

NOW FOR A LITTLE PROBLEM.

HERE IS THE PROBLEM:

IN THIS WORKSPACE IS STORED A FUNCTION 'ESCAPE' THAT DOES SOME INTERESTING COMPUTATIONS.
HOWEVER, THE FUNCTION DOES NOT WORK PROPERLY. USING THE EDITING FACILITIES THAT YOU HAVE
LEARNED, I WANT YOU TO CORRECT 'ESCAPE'.

CORRECT THE FOLLOWING ERRORS:

1. LIST THE ENTIRE FUNCTION AS IT STANDS SO YOU CAN SEE THE ERRORS I REFER TO.
2. LINE [22] SHOULD READ: VELOCMKS←(2×G×M÷R)*0.5 FIX IT.
3. INSERT A LINE BETWEEN PRESENT LINES [6] AND [7] THAT OUTPUTS:
   'IT'S THE VELOCITY REQUIRED OF A BODY FOR IT TO OVERCOME GRAVITY AND REMAIN IN ORBIT.'
4. 'NINE' IS MISSPELLED IN PRESENT LINE [1]. FIX IT.
5. 'BODY' IS MISSPELLED IN PRESENT LINE[20]. FIX IT.
6. MAKE 'G' LOCAL TO 'ESCAPE' BY ADDING IT TO THE HEADER LINE.

WHEN YOU FEEL THAT YOU HAVE CORRECTED THE FUNCTION SATISFACTORILY, TRY EXECUTING IT
TO

```
 SEE IF I
         T SEEMS TO WORK.  WHEN YOU ARE SATISFIED, I'D LIKE TO PERFO
RM A FEW CHECKS MYSELF.
SO, WHEN YOU ARE READY TO HAVE ME CHECK IT.  TYPE 'READY'.

GOOD LUCK!
      ∇ESCAPE[□]
    ∇ ESCAPE;R;M;VELOCENG
[1]   'THIS FUNCTION WILL COMPUTE THE ''ESCAPE VELOCITY'' FOR ANY OF T
HE NINT PLANETS AND EARTH''S MOON.'
[2]   ''
[3]   'DO YOU KNOW WHAT ''ESCAPE VELOCITY'' IS?'
[4]   ''
[5]   →ESC1×ι'Y'=1ρ□,' '
[6]   ''
[7]  ESC1:G←6.7E¯11
[8]   ''
[9]   'HERE ARE THE BODIES CONSIDERED:'
[10]  ''
[11]  '      MERCURY0      JUPITER'
[12]  '      VENUS         SATURN'
[13]  '      EARTH        0URANUS'
[14]  '      MOON          NEPTUNE'
[15]  '      MARS          PLUTO'
[16]  ''
[17] ESC2:'FROM WHICH BODY WOULD YOU LIKE TO LIFT OFF?'
[18]  ''
[19]  BODY←□
[20]  R←RADIUS[BDY]
[21]  M←MASS[BODY]
[22]  VELOCMKS←(G×M÷R)*0.5
[23]  VELOCENG←2.3693182×VELOCMKS
[24]  ''
[25]  'ESCAPE VELOCITY IS ';VELOCENG;' MPH.'
[26]  ''
[27]  'MORE?'
[28]  ''
[29]  →ESC2×ι'Y'=1ρ□,' '
[30]  ''
[31]  'HAPPY LANDINGS!'
    ∇
[32]  [22]
[22]  VELOMKS←(2×G×M÷R)*0.5
[23]  ∇
      ∇ESCAPE[6.5]
[6.5] 'IT''S THE VELOCITY REQUIRED OF A BODY FOR IT TO OVERCOME GRAVI
[6.6] [1□75]
DEFN ERROR
[6.6] [1□75]
           ^
[1]   [1□60]
[1]   'THIS FUNCTION WILL COMPUTE THE ''ESCAPE VELOCITY'' FOR ANY OF T
HE NINT PLANETS AND EARTH''S MOON.'
[1]   )WIDTH
      ∨

[1]   ∇
      )LOAD
      ∨
      ∇ESCAPE[□]∇
CHARACTER ERROR
      ∇ESCAPE[□]∇
    ∇ ESCAPE;R;M;VELOCENG
[1]   'THIS FUNCTION WILL COMPUTE THE ''ESCAPE VELOCITY'' FOR ANY OF T
HE NINT PLANETS AND EARTH''S MOON.'
[2
```

```
      ]
          ''
[3]       'DO YOU KNOW WHAT ''ESCAPE VELOCITY'' IS?'
[4]       ''
[5]       →ESC1×ι'Y'=1ρ⍞,' '
[6]       ''
[7]       'IT''S THE VELOCITY REQUIRED OF A BODY FOR IT TO OVERCOME GRAVI
Y AND REMAIN IN ORBIT.'
[8]     ESC1:G←6.7E¯11
[9]       ''
[10]      'HERE ARE THE BODIES CONSIDERED:'
[11]      ''
[12]      '      MERCURY       JUPITER'
[13]      '      VENUS         SATURN'
[14]      '      EARTH         URANUS'
[15]      '      MOON          NEPTUNE'
[16]      '      MARS          PLUTO'
[17]      ''
[18]    ESC2:'FROM WHICH BODY WOULD YOU LIKE TO LIFT OFF?'
[19]      ''
[20]      BODY←⍞
[21]      R←RADIUS[BDY]
[22]      M←MASS[BODY]
[23]      VELOMKS←(2×G×M÷R)*0.5
[24]      VELOCENG←2.3693182×VELOCMKS
[25]      ''
[26]      'ESCAPE VELOCITY IS ';VELOCENG;' MPH.'
[27]      ''
[28]      'MORE?'
[29]      ''
[30]      →ESC2×ι'Y'=1ρ⍞,' '
[31]      ''
[32]      'HAPPY LANDINGS!'
        ∇
          ∇ESCAPR
                ∨
                E[23]
[23]      [23⌊6]
[23]      VELOMKS←(2×G×M÷R)*0.5
              /1
[23]      VELOMKS←(2×G×M÷R)*0.5
[24]      [23⌈14]
[23]    VELOMKS←(2×G×M÷R)*0.5
            1
[23]    VELOCMKS←(2×G×M÷R)*0.5
[24]      ∇
          )WIDTH 120
    WAS 70
          ∇ESCAPE[1⌈70]
[1]       'THIS FUNCTION WILL COMPUTE THE ''ESCAPE VELOCITY'' FOR ANY OF

    DEFN ERROR
    [1]   [1⌈70)
    DEFN ERROR
    [1]   [1⌊70)
                ^
    [1]   [1⌈70]
    [1]   'THIS FUNCTION WILL COMPUTE THE ''ESCAPE VELOCITY'' FOR ANY OF

    [1]   'THIS FUNCTION WILL COMPUTE THE ''ESCAPE VELOCITY'' FOR ANY OF
    [2]   [1⌈70]
    [1]   'THIS FUNCTION WILL COMPUTE THE ''ESCAPE VELOCITY'' FOR ANY OF T
          0
    [1]   'THIS FUNCTION WILL COMPUTE THE ''ESCAPE VELOCITY'' FOR ANY OF T
7-  [2]
```

```
      [21⎕20]
[21]  R←RADIUS[BDY]
                  1
[21]  R←RADIUS[BODY]
[22]  [0]
[0]
[0]     ∇
      ∇ESCAPE[0⎕]
    ∇ ESCAPE;R;M;VELOCENG
[0]     ∇
      ∇ESCAPE[0⎕30]
[0]  ESCAPE;R;M;VELOCENG
                        2
[0]  ESCAPE;R;M;VELOCENG;G
[1]    [0]
[0]     ∇
      ∇ESCAPE[⎕]∇
    ∇ ESCAPE;R;M;VELOCENG;G
[1]    'THIS FUNCTION WILL COMPUTE THE ''ESCAPE VELOCITY'' FOR ANY OF
[2]    ''
[3]    'DO YOU KNOW WHAT ''ESCAPE VELOCITY'' IS?'
[4]    ''
[5]    →ESC1×ι'Y'=
      )WIDTH 70
WAS 120
      ∇ESCAPE[⎕]∇
    ∇ ESCAPE;R;M;VELOCENG;G
[1]    'THIS FUNCTION WILL COMPUTE THE ''ESCAPE VELOCITY'' FOR ANY OF
HE NINE PLANETS AND EARTH''S MOON.'
[2]    ''
[3]    'DO YOU KNOW WHAT ''ESCAPE VELOCITY'' IS?'
[4]    ''
[5]    →ESC1×ι'Y'=1ρ⍞,' '
[6]    ''
[7]    'IT''S THE VELOCITY REQUIRED OF A BODY FOR IT TO OVERCOME GRAVI
Y AND REMAIN IN ORBIT.'
[8]  ESC1:G←6.7E¯11
[9]    ''
[10]   'HERE ARE THE BODIES CONSIDERED:'
[11]   ''
[12]   '      MERCURY       JUPITER'
[13]   '      VENUS         SATURN'
[14]   '      EARTH         URANUS'
[15]   '      MOON          NEPTUNE'
[16]   '      MARS          PLUTO'
[17]   ''
[18] ESC2:'FROM WHICH BODY WOULD YOU LIKE TO LIFT OFF?'
[19]   ''
[20]   BODY←⍞
[21]   R←RADIUS[BODY]
[22]   M←MASS[BODY]
[23]   VELOCMKS←(2×G×M÷R)*0.5
[24]   VELOCENG←2.3693182×VELOCMKS
[25]   ''
[26]   'ESCAPE VELOCITY IS ';VELOCENG;' MPH.'
[27]   ''
[28]   'MORE?'
[29]   ''
[30]   →ESC2×ι'Y'=1ρ⍞,' '
[31]   ''
[32]   'HAPPY LANDINGS!'
```

```
     ∇
        ESCAPE
THIS FUNCTION WILL COMPUTE THE 'ESCAPE VELOCITY' FOR ANY OF THE NINE
LANETS AND EARTH'S MOON.

DO YOU KNOW WHAT 'ESCAPE VELOCITY' IS?

0

IT'S THE VELOCITY REQUIRED OF A BODY FOR IT TO OVERCOME GRAVITY AND R
MAIN IN ORBIT.

HERE ARE THE BODIES CONSIDERED:

      MERCURY        JUPITER
      VENUS          SATURN
      EARTH          URANUS
      MOON           NEPTUNE
      MARS           PLUTO

FROM WHICH BODY WOULD YOU LIKE TO LIFT OFF?

□:
       EARTH

ESCAPE VELOCITY IS 18746.55183 MPH.

MORE?

0

HAPPY LANDINGS!
       READY
LOOKS SATISFACTORY TO THE CHECK I'VE GIVEN IT.

YOU MAY WANT TO FOOL AROUND WITH THE 'ESCAPE' FUNCTION.

DO YOU WANT TO?

1
YES OR NO?
YES

OKAY, GO AHEAD.  TYPE 'DONE' WHEN YOU ARE THROUGH.
       ∇ESCAPE[1□15]
[1]    'THIS FUNCTION WILL COMPUTE THE ''ESCAPE VELOCITY'' FOR ANY OF
HE NINE PLANETS AND EARTH''S MOON.'
[1]
[1]    [0□15]
[0]  ESCAPE;R;M;VELOCENG;G
                          //
[0]  ESCAPE;R;M;VELOCENG
[1]  [7 ]
[7]
[7]    ∇
       ∇ESCAPE[7□]∇
[7]    'IT''S THE VELOCITY REQUIRED OF A BODY FOR IT TO OVERCOME GRAVI
Y AND REMAIN IN ORBIT.'
       ∇ESCAPE[7□]
[7]    'IT''S THE VELOCITY REQUIRED OF A BODY FOR IT TO OVERCOME GRAVI
Y AND REMAIN IN ORBIT.'
[7]
[7]
[7]
[7]    ∇
```

```
      G
      ∨
      ∇ESCAPE
[33]  [7]
[7]
[7]   ∇
      ∇ESCAPE[7]
[7]
[7] 0
[7]   ∨
      'BARST MET JE ZOOI'
BARST MET JE ZOOI
      DONE
```

THAT'S IT FOR FUNCTION EDITING!

THE NEXT MODULE IS ¨46 *LESSON17*¨ *WHICH DEALS WITH THESE PRIMITIVE FUNCTIONS:* ? , ∈ ⍑ ⊥

```
      )LOAD 46 LESSON17
SAVED  18.31.48 10/06/75
      )WIDTH 70
WAS 120
      START
```


LESSON 17

IN THIS MODULE WE SHALL STUDY 7
MORE PRIMITIVE FUNCTIONS. THEY ARE:

MONADIC: ? ,
DYADIC: ? , ∈ ⊥ ⊤

DO YOU WISH TO SKIP ANY?

NO

HERE'S A FUN ONE!

EXECUTE: ?10 AT LEAST 4 TIMES, AND LOOK AT THE OUTPUT.

WHEN YOU'VE DONE THAT, AND WANT TO CONTINUE, TYPE: "CONT1"

```
      ?10
2
      ?10
8
      ?10
5
      ?10
6
      ?10
3
      CONT1
```

DID THE OUTPUT APPEAR TO HAVE A RANDOM PATTERN?

YES

THE FUNCTION "?N" DOES INDEED GENERATE RANDOM INTEGERS!
IT YIELDS AN INTEGER CHOSEN RANDOMLY FROM THE SET "⍳N".

THIS FUNCTION IS SCALAR - THIS MEANS THAT ARRAYS OF
INTEGERS MAY BE USED AS ITS ARGUMENT.

TO SEE WHAT I MEAN, DO THIS:

1. DEFINE A VECTOR "V" SPECIFIED AS 6 6 (I.E., V←6 6).
2. EXECUTE: ?V FIVE OR SIX TIMES, AND EXAMINE THE RESULTS.

1- WHEN YOU'VE DONE THAT, TYPE: "CONT2"

```
      V←6 6 6
      ?V
1 5 5
      ?V
6 3 4
      ?V
5 1 1
      ?V
4 5 1
      ?V
3 1 3
      CONT2
```

YOU WILL HAVE NOTICED THAT 2 RANDOM NUMBERS WERE GENERATED EACH TIME YOU EXECUTED "?V". YOU MIGHT INTERPRET THIS AS ANALOGOUS TO THROWING A PAIR OF DICE.

SO MUCH SO, IN FACT, THAT THIS FUNCTION ('?') HAS BEEN GIVEN THE NAME "ROLL".

NOTICE THAT THE RANDOM SELECTION WAS MADE <u>WITH REPLACEMENT</u> EACH TIME - WE SHALL SEE LATER HOW WE CAN GENERATE RANDOM INTEGERS <u>WITHOUT</u> REPLACEMENT FROM A SET.

THE MONADIC COMMA (CALLED "RAVEL") IS ONE OF THE SIMPLEST FUNCTIONS IN APL - IT HAS ONE PURPOSE ONLY: IT WILL RESHAPE <u>ANY</u> ARGUMENT INTO A VECTOR FORMAT!

LEST YOU THINK THIS TRIVIAL, CONSIDER THE PROBLEM OF DEALING WITH DATA OF UNKNOWN DIMENSIONS. MERELY PRECEDING THESE DATA WITH A COMMA WILL RESHAPE THEM INTO A FORMAT YOU KNOW - THE VECTOR.

IT IS ALSO USEFUL FOR CHANGING A SCALAR INTO A ONE-COMPONENT VECTOR (ESPECIALLY SO IF YOU HAVE A SINGLE QUANTITY AS THE LEFT ARGUMENT OF DYADIC "⍳", SINCE THIS <u>MUST</u> BE A VECTOR!).

THUS, REGARDLESS OF ITS FORMER SHAPE, THE NEW SHAPE OF THE RAVEL OF DATA IS THAT OF A VECTOR.

NOW, ENTER

```
      AL←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      M←2 3 ⍴AL
      AR←2 3 2⍴AL
```

THEN, IN SEQUENCE FIND

```
      ⍴AL
      ⍴,AL
      ⍴M
      ⍴,M
      M
      ,M
      ⍴AR
      ⍴,AR .
```

TYPE CONT7 WHEN YOU HAVE FINISHED THIS.

```
       AL←'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
       M←2 3⍴AL
       AR←2 3 2⍴AL
       ⍴AL
```

```
      ρ,AL
26
      ρM
2 3
      ρ,M
6
      M
ABC
DEF
      ,M
ABCDEF
      ρAR
2 3 2
      ρAL
26
      ρ,AR
12
      CONT7
```

YOU SHOULD HAVE OBSERVED THAT THE INSERTION OF THE FUNCTION SYMBOL , PRODUCED A VECTOR. THAT IS BASICALLY ALL THAT THERE IS TO THE RAVEL FUNCTION. HOWEVER, BEFORE WE LEAVE IT, TRY ONE MORE SITUATION. DEFINE

```
     S←'A'
     V←,'A'
```

THEN FIND

```
     ρS   AND
     ρV.
```

OBSERVE THE OUTCOME THEN TYPE CONT8.

```
      2←'A'
SYNTAX ERROR
      2←'A'
      ^
      S←'A'
      V←,'A'

'HELLO'
HELLO
      ρS

      ρV
1
      CONT8
```

NOTICE THAT ρS WAS A NULL VECTOR! THIS WAS SO BECAUSE S IS A SCALAR, AND YOU WILL RECALL THAT A SCALAR HAS NO DIMENSION.

HOWEVER, ρV GENERATED A 1 BECAUSE V IS A 1-COMPONENT VECTOR.

RECALL THAT WHEN THE MONADIC "?" FUNCTION WAS USED WITH AN ARRAY ARGUMENT, I SAID IT GENERATED RANDOM INTEGERS WITH REPLACEMENT.

FOR EXAMPLE:

```
      ?6ρ5
4 3 5 5 3 1
```

3-

HOWEVER, THERE ARE MANY TIMES WHEN IT IS DESIRABLE TO

GENERATE SETS OF RANDOM NUMBERS WITHOUT REPLACEMENT - THAT IS, ONCE A NUMBER HAS BEEN SELECTED, IT WILL NOT RE-APPEAR IN THE SET.

THE DYADIC USE OF THE "?" FUNCTION DOES JUST THAT!
EXECUTE: `3?9` *SEVERAL TIMES, AND EXAMINE THE OUTPUT.*

WHEN YOU'VE DONE THAT, TYPE: `"CONT3"`

```
      3?9
4 6 3
      3?9
1 3 7
      3?9
9 3 7
      3⍴
       ⍳
       ?9
4 3 1
      3?9
6 7 1
      9?9
6 9 8 5 7 2 1 3 4
      10?9
DOMAIN ERROR
      10?9
      ∧
      CONT3
```

THE GENERAL FORM OF THE DYADIC "?" IS:

```
       M?N
```

WHERE: 1. *BOTH ARGUMENTS ARE EITHER SCALARS, OR ONE-COMPONENT ARRAYS,*
AND: 2. *THEY ARE BOTH POSITIVE INTEGERS.*

LET'S TRY IT.

SPECIFY `V←1 1⍴9` *AND THEN EXECUTE* `3?V` *SEVERAL TIMES.*

WHEN YOU HAVE DONE THAT, TYPE: `"CONT4"`

```
      V←1 1⍴9
      3?9
5 7 8
      3?9
6 8 5
      V←2 2⍴6
      3?V
RANK ERROR
      3?V
      ∧
      V←1 1⍴9
      3?V
7 9 1
      3?V
1 5 8
      CONT4
```

NOT ONLY MUST BOTH ARGUMENTS BE SCALARS OR 1 ELEMENT ARRAYS, BUT ALSO THE LEFT ARGUMENT MUST BE LESS THAN OR EQUAL TO THE RIGHT ARGUMENT. AFTER ALL WHAT WOULD IT MEAN TO SELECT WITHOUT REPLACEMENT 5 RANDOM INTEGERS FROM THE SET OF 4.

THAT IS WHAT 5?4 WOULD BE ASKING FOR. ENTER SUCH AN EXPRESSION AND DETERMINE THE ERROR MESSAGE. TO CONTINUE TYPE: "CONT5" .

```
      CONT5
```

IF A DECK OF CARDS WERE NUMBERED SEQUENTIALLY FROM 1 TO 52, A 5-CARD HAND CAN BE RANDOMLY DEALT WITH THE EXPRESSION:

```
      5?52
```

PERHAPS THIS EXPLAINS WHY THE NAME OF THE DYADIC "?" IS "DEAL"!

WHERE THE LEFT ARGUMENT IS IDENTICAL TO THE RIGHT ARGUMENT, THE RESULT WILL BE A RANDOM ORDERING OF THE SET: ⍳(EITHER ARGUMENT). THIS IS CALLED A "RANDOM PERMUTATION VECTOR".

AT THIS POINT, IF YOU WISH TO TAKE A MINI-QUIZ ON "ROLL" AND "DEAL", TYPE: MINIQUIZ

TO CONTINUE TO A NEW TOPIC, TYPE: CONT6

```
      MINIQUIZ
```

THERE ARE 3 QUESTIONS ON "?".

1. THE EXPRESSION: 10+?5 5 GENERATES:

   A. A PAIR OF RANDOM NUMBERS BETWEEN 1 AND 5.
   B. A PAIR OF RANDOM NUMBERS BETWEEN 10 AND 15 WITHOUT REPLACEMENT.
   C. A RANDOM NUMBER BETWEEN 10 AND 15 WITH REPLACEMENT.
   D. A PAIR OF RANDOM NUMBERS BETWEEN 10 AND 15 WITH REPLACEMENT.
   E. NONE OF THE ABOVE.

TYPE THE LETTER REPRESENTING THE ANSWER YOU SELECT:

```
B
```

NO, MONADIC "?" GENERATES RANDOM NUMBERS WITH REPLACEMENT. THE EXPRESSION GENERATES A PAIR OF RANDOM NUMBERS BETWEEN 11 AND 15 WITH REPLACEMENT. (CORRECT ANSWER WAS E.)

2. THE EXPRESSION: 3?5 5 GENERATES:

   A. 3 RANDOM NUMBERS FROM THE SET ⍳5 WITH REPLACEMENT.
   B. 3 RANDOM NUMBERS FROM THE SET ⍳5 WITHOUT REPLACEMENT.
   C. "SYNTAX ERROR" MESSAGE.
   D. "RANK ERROR" MESSAGE.
   E. NONE OF THE ABOVE.

TYPE THE LETTER OF THE ANSWER OF YOUR CHOICE:

```
D
```

CORRECT!

3. WHICH OF THE FOLLOWING EXPRESSIONS WILL GENERATE A PAIR OF INTEGERS BETWEEN ¯10 AND 10 (INCLUSIVE) WITHOUT REPLACEMENT:

   A. ¯10+2?20
   B. ¯11+?20 20
   C. ¯11+?21 21
   D. ¯11+2?21
   E. NONE OF THE ABOVE.

ENTER THE LETTER OF YOUR ANSWER:

D
CORRECT!

17

CATENATE

THIS FUNCTION, WHICH IS THE DYADIC USE OF THE COMMA, WILL JOIN ITS ARGUMENTS.

FOR EXAMPLE:

```
ASSUME THAT    V1←⍳5
AND            V2←8 4 0 2

THEN           V1,V2  ←→ 1 2 3 4 5 8 4 0 2
AND            V2,V1  ←→ 8 4 0 2 1 2 3 4 5
```

THE ARGUMENTS MUST BOTH BE OF THE SAME DATA TYPE. THAT IS, LITERAL DATA MAY BE CATENATED TO ONLY LITERAL DATA, AND NUMERIC DATA CAN BE CATENATED TO ONLY NUMERIC DATA.

THE CATENATE FUNCTION WILL OPERATE ON DATA OF ANY DIMENSION, BUT IN THIS MODULE WE SHALL RESTRICT OUR DISCUSSION TO SCALARS AND VECTORS.

THE GENERAL FORM:

```
      X,Y
```

YIELDS A VECTOR WHOSE COMPONENTS ARE THE ELEMENTS OF "X" FOLLOWED BY THE ELEMENTS OF "Y" REGARDLESS OF WHETHER "X" OR "Y" IS SCALAR, A 1-COMPONENT VECTOR, OR A VECTOR OF GREATER LENGTH.

SUPPOSE THE VARIABLE "SCORES" CONTAINED THE RESULTS OF A SERIES OF TESTS, AND THAT:

```
      SCORES
85 81 92 86
```

THEN TO ADD ANOTHER SCORE (SAY: 95) YOU'D ENTER:

```
      SCORES,95
85 81 92 86 95
```

AND TO SAVE "SCORES" FOR FUTURE REFERENCE:

```
      SCORES←SCORES,95
```

EXPERIMENT WITH CATENATE FOR YOURSELF, AND TRY USING SOME LITERAL DATA CATENATIONS.

WHEN YOU'RE READY TO CONTINUE, TYPE: CONT11

```
      TEKST←'ALLE '
      TEKST←TEKST,'GEKHEID OP EEN '
      TEKST←TEKST,'STOKJE'
      TEKST
ALLE GEKHEID OP EEN STOKJE
      V←⍳4,¯2↑⍳4
RANK ERROR
      V←⍳4,¯2↑⍳4
        ^
      V←⍳4,1↑⍳4
```

```
RANK ERROR
      V←⍳4,1↑⍳4
        ^
      V←⍳4,⍳7
RANK ERROR
      V←⍳4,⍳7
        ^
      V←⍳4
      V←V,¯2↑⍳4
      V
1 2 3 4 3 4
      CONT11
```

THE DYADIC COMMA REPRESENTS THE FUNCTION OF CATENATION. THUS, THE EXPRESSIONS:

```
      0 1 2+3
```

AND

```
      0 1,2+3
```

ARE NOT IDENTICAL!

YOU MAY TRY THEM FOR YOURSELF IF YOU WISH, AND, WHEN YOU WANT TO CONTINUE, TYPE: CONT12

IF YOU'D LIKE SOME FURTHER COMMENTS ON CATENATION, TYPE: HELP1

```
      0 1 2+3
3 4 5
      0 1 ,1
           v
           2-
            v
            +3
0 1 5
      HELP1
```

THE EXPRESSION: 0 1 2+3 *YIELDS THE 3-COMPONENT VECTOR:* 3 4 5
WHICH IS THE RESULT OF ADDING 3 *TO EACH OF THE ELEMENTS* 0 1 2.

THE EXPRESSION: 0 1,2+3 *YIELDS THE 3-COMPONENT VECTOR:* 0 1 5
WHICH IS THE RESULT OF:

1. *THE SUM OF* 2 *AND* 3
2. *THE CATENATION OF THE* 2*-COMPONENT VECTOR* 0 1 *TO THE SUM OF AND* 3

(REMEMBER THE RIGHT-TO-LEFT SCAN!)

NOW WE'LL TRY SOME EASY PROBLEMS!

ENTER THE APL EVALUATION OF THIS EXPRESSION:

```
     6,5 3 1 10 1+2
⎕:
      6 5 3 1 10 3
```

NO, THE ANSWER IS 6 7 5 3 12 3
HOW ABOUT THIS ONE?

```
     10 1 8,4+4 3 9
⎕:
      10 1 8 8 7 13
```

FINE!
SHALL WE TRY AGAIN?

```
YES

HOW ABOUT THIS ONE?
      9 7 8 8,1+7 8 2
□:
       9 7 8 8 8 9 3
FINE!
SHALL WE TRY AGAIN?
NO


THAT'S IT FOR CATENATION!
```

THE MEMBERSHIP FUNCTION

OFTEN WE SEARCH TO DETERMINE IF A GIVEN ITEM "X" IS PRESENT ANYWHERE WITHIN A SET OF ITEMS, "Y". WE ARE NOT CONCERNED NECESSARILY ABOUT WHERE IN THE SET BUT JUST IS IT THERE - YES OR NO?

THE DYADIC FUNCTION MEMBERSHIP WITH THE SYMBOL "∈" (UPPER CASE E) PROVIDES THE ANSWER TO THAT QUESTION. LET'S SEE HOW. DEFINE

```
      V←'CAT'
      AL←'ABCD'
```

THEN TYPE SUCCESSIVELY

```
      'C'∈AL
      'CA'∈AL
      'T'∈AL
      V∈AL
```

THEN TYPE CONT9 TO CARRY ON.

```
       V←'CAT'
       AL←'ABCD';'C'∈AL
ABCD1
       AL←'ABCD
SYNTAX ERROR
       AL←'ABCD
          ^
       AL←'ABCD'
       'C'∈AL
1
       'CA'∈AL
1 1
       'T'∈AL
0
       V∈AL
1 1 0
       CONT9
```

YOU SHOULD HAVE NOTICED THAT YOU RECEIVED A SET OF 1'S AND 0'S ON ALL THESE APPLICATIONS OF ∈ . EACH OF THE COMPONENTS OF THE LEFT ARGUMENT PRODUCED A 1 OR A 0 DEPENDENT ON WHETHER IT IS PRESENT OR NOT IN THE RIGHT ARGUMENT. IF WE HAVE

```
      A∈B
```

THIS CAN BE READ AS ' ARE THE ELEMENT(S) OF A FOUND IN THE SET B ?'. FOR EACH ELEMENT OF A PRESENT, WE RECEIVE A 1 AND FOR EACH ELEMENT NOT PRESENT WE RECEIVE A 0.

THE SHAPES OF THE ARGUMENTS NEED NOT BE THE SAME.

HOWEVER, THE SHAPE OF THE RESULT IS THE SAME AS THE LEFT ARGUMENT. LET'S SEE. WE ALREADY HAVE AL DEFINED, NOW DEFINE

```
      M←3 4 ⍴'TALLDADS'
```

AND THEN ASK FOR

```
                  AL∈M        AND
                  M∈AL.
```

TRY SOME OTHER VARIABLES WITH DIFFERENT SHAPES. TYPE CONT10 TO CONTINUE .

```
      M←3 4⍴'TALLDADS'
      A[
       ∨
       L∈M
1 0 0 1
      M∈AL
0 1 0 0
1 1 1 0
0 1 0 0
      CONT19
           ∨
           0
```

A USEFUL APPLICATION OF ∈ OCCURS WHEN ONE WISHES TO DETECT THE OCCURENCES OF A SPECIFIC COMPONENT IN A VECTOR. THUS IF

```
      ALP←'SMITH, J. A.'
```

AND

```
      R←ALP∈','
```

THEN R IS A LOGICAL VECTOR WITH ONES WHEREVER A COMMA APPEARS IN THE CHARACTER STRING ALP. NOW ACTUALLY ENTER THESE TWO EXPRESSIONS AND CHECK IT OUT. AFTER THAT SPECIFY R TO BE THE LOGICAL VECTOR INDICATING WHERE THE PERIODS OCCUR IN THE VECTOR ALP . TYPE "READY" WHEN YOU HAVE COMPLETED THAT AND YOU WISH TO BE CHECKED.

```
      ALP←'SMITH, J. A.'
      R←ALP∈','
      R
0 0 0 0 0 1 0 0 0 0 0 0
      R←⍳⍴R/(R)
      R
1
      R←6
      READY
```

NO. THE CORRECT EXPRESSION IS:

```
      R←ALP∈'.'
```

WOULD YOU LIKE A SMALL DRILL ON MEMBERSHIP?
YES

WHAT IS THE VALUE OF THE EXPRESSION:

```
      3 5∈14 4 5 6 8 9 13 7 13 5 7 9 8 5 3

⎕:
      1

OK, NO
ATTENTION
READY[9]
      →9
⎕:
```

```
      1 1
NO, THE VALUE FOR 14 4 5 6 8 9 13 7 13 5 7 9 8 5 3∈3 5 IS:

0 0 1 0 0 0 0 0 0 1 0 0 0 1 1

LET'S TRY ONCE MORE

WHAT IS THE VALUE OF THE EXPRESSION:

      5 4 3 5 4∈9

⎕:
      0 0 0 0 0

OK, NOW HOW ABOUT THE VALUE OF:

      9∈5 4 3 5 4

⎕:
      0
FINE!

SO MUCH FOR MEMBERSHIP
```

DECODE

THE EVALUATION OF A POLYNOMIAL IS THE BASIS FOR MANY COMMON ACTIVITIES. SUCH DIVERSE ITEMS AS:

1. THE DETERMINATION OF THE DECIMAL EQUIVALENT OF NUMBERS IN OTHER BASES.
2. THE CONVERSION OF MIXED UNIT MEASUREMENTS TO THEIR COMMON UNIT.

ARE RELATED TO POLYNOMIAL EVALUATION.

THE EXPRESSION:

```
      Y←(X*2)+(2×X)+3
```

IS AN EXAMPLE OF A SIMPLE POLYNOMIAL IN THE UNKNOWN "X".

TO EVALUATE THIS POLYNOMIAL, WE SHALL NEED TO KNOW:

1. THAT VALUE OF "X" FOR WHICH THE POLYNOMIAL IS TO BE EVALUATED.
2. THE COEFFICIENTS OF THE POLYNOMIAL.

IN ORDER TO ACCOMPLISH THIS IN APL, WE HAVE THE <u>DECODE</u> FUNCTION, WHOSE SYMBOL IS ⊥ (UPPER CASE B). ITS GENERAL FORM IS:

```
      A⊥B
```

WHERE: A REPRESENTS THE VALUE OF "X"
AND: B IS A VECTOR REPRESENTING THE COEFFICIENTS IN DESCENDING ORDER.

THUS, TO EVALUATE OUR POLYNOMIAL FOR X←10 WE ENTER:

```
      10⊥1 2 3   (OR: 10⊥⍳3  IN THIS CASE.)
```

<u>IMPORTANT</u>!

IF ONE OF THE EXPONENTS OF THE UNKNOWN IS <u>NOT</u> PRESENT IN THE

POLYNOMIAL, IT IS CONSIDERED TO HAVE A 0 COEFFICIENT.

THUS, TO EVALUATE: (5×X*4)+(X*2)-23 *FOR* X←6 *WE WOULD ENTER:*

```
      6⊥5 0 1 0 ¯23
```

LET'S TRY SOME!

ENTER: 10⊥1 2 3

```
□:
      10⊥1 2 3
123
```

THAT'S A SIMPLE ONE. YOUR ANSWER SHOULD BE 123

NOW ENTER: 8⊥1 2 3

```
□:
      8⊥1 2 3
83
```

FINE!
HOW ABOUT SOME DRILL?

YES

ENTER THE APL EXPRESSION WHICH WILL EVALUATE:

```
     (7×X*3) + (2.2×X) + ¯16.9
```

WITH X←1

```
□:
      1⊥7 0 2,
              v
             .2 ¯16.9
```

CORRECT!

WOULD YOU LIKE ANOTHER?
NO

WE HAVE BEEN INTERPRETING THE EXPRESSION 8⊥1 2 3 *AS THE EVALUATION OF A POLYNOMIAL FOR* X←8.

ANOTHER WAY TO LOOK AT IT IS THAT IT ANSWERS THE QUESTION:

'WHAT IS THE DECIMAL EQUIVALENT OF OCTAL 123*?'*

SO WE SEE THAT ONE OF THE USES FOR DECODE IS TO DETERMINE THE DECIMAL EQUIVALENT OF INTEGER NUMBERS IN A DIFFERENT BASE (OR RADIX).

GIVEN THE GENERAL FORM:

```
      A⊥B
```

DECODE YIELDS THE DECIMAL EQUIVALENT AS A SCALAR WHEN:

A IS THE BASE
AND: B IS A VECTOR WHOSE COMPONENTS ARE THE DIGITS OF THE NUMBER.

FOR EXAMPLE:

TO

FIND THE DECIMAL EQUIVALENT OF THE OCTAL NUMBER 245 (245 TO BASE 8
WE ENTER:

```
      8⊥2 4 5
```

LET'S SEE IF YOU'VE GOT THAT.
PLEASE TYPE: QU1

```
      QU1
```

ENTER THE EXPRESSION FOR NUMBER 2 TO BASE 4
```
[]:
      4⊥2
2
```
FINE!
WANT ANOTHER?
YES

ENTER THE EXPRESSION FOR NUMBER 11111 TO BASE 2
```
[]:
      2⊥1 1 1 1 1
31
```
FINE!
WANT ANOTHER?
NO

DECODE COULD ALSO BE USED, FOR EXAMPLE, TO DETERMINE THE NUMBER OF SECONDS IN, SAY, 1 HOUR, 20 MINUTES, AND 15 SECONDS.

YOU WOULD ENTER:

```
      24 60 60⊥1 20 15
```

TRY IT OUT NOW.

WHEN YOU'VE FINISHED, TYPE: CONT13

```
      24 60 60⊥1 20 15
4815
      CONT13
```
YOUR ANSWER SHOULD HAVE BEEN 4815 SECONDS. IT IS IMPORTANT TO NOTE THAT IN A⊥B IF A IS A VECTOR, THEN

```
          (ρA)=ρB.
```

THE COMPONENTS PLACED IN IT DEPEND UPON THE RELATIONSHIP BETWEEN THE UNITS. IN OUR PREVIOUS ILLUSTRATION READING FROM RIGHT TO LEFT AND MOVING FROM THE SMALLEST UNIT TO THE LARGEST, WE HAVE

60 SECONDS IN A MINUTE,
60 MINUTES IN AN HOUR, AND
24 HOURS IN A DAY.

HENCE THE VECTOR LEFT ARGUMENT OF 24 60 60. NOW IF YOU DON'T KNOW THE RELATION BEYOND THE LARGEST, A ZERO WILL DO. THUS

```
          0 60 60 ⊥1 20 15
```

WILL ALSO PRODUCE THE SAME RESULT. TRY IT. THEN TYPE CONT14 TO CONTINUE.

```
      CONT14
```

THIS APPLICATION OF THE DECODE FUNCTION WITH A VECTOR LEFT ARGUMENT

ALLOWS YOU TO DETERMINE THE "COMMON UNIT VALUE" OF A SET OF NUMBERS WITH EACH GIVEN IN DIFFERENT UNITS. (WE SHALL ASSUME THAT THEY DO, IN FACT, HAVE A COMMON UNIT!)

THE LEFT ARGUMENT IS A VECTOR WHICH INDICATES THE RELATIONSHIP OF (OR AMONG) THE ELEMENTS OF THE RIGHT ARGUMENT. THE RIGHT-MOST COMPONENT OF THIS VECTOR ALWAYS DEFINES THE NUMBER OF THE SMALLEST UNITS IN THE NEXT LARGER, THEN THE NEXT-TO-RIGHT-MOST DEFINES THE NUMBER OF UNITS IN THE 3RD, AND SO ON.

I REALIZE THAT THIS DEFINITION MAY NOT BE CLEAR WITHOUT AN EXAMPLE, SO LET'S DETERMINE THE VECTOR LEFT ARGUMENT WHICH INDICATES THE NUMBER OF INCHES IN A VECTOR RIGHT ARGUMENT WHICH REPRESENTS YARDS, FEET, AND INCHES.

FOR EXAMPLE: 3 YARDS 2 FEET 8 INCHES

THERE ARE 12 INCHES IN ONE FOOT, SO THE RIGHT-MOST COMPONENT WILL BE 12.

THERE ARE 3 FEET IN ONE YARD, SO THE NEXT-TO-RIGHT-MOST COMPONENT WILL BE 3.

THERE ARE 1760 YARDS IN ONE MILE, SO THE LEFT-MOST COMPONENT WILL BE 1760. IF YOU DON'T CARE ABOUT MILES, THIS COMPONENT COULD ALSO BE ZERO - EITHER WAY WILL WORK! (WHICH IS JUST ANOTHER WAY OF SAYING THAT THE LEFT-MOST COMPONENT CAN ALWAYS BE 0.)

NOW FOR SOME PROBLEMS!

ENTER THE APL EXPRESSION (USING THE DECODE FUNCTION!) WHICH DETERMINES THE NUMBER OF INCHES IN 3 YARDS, 2 FEET, 8 INCHES.

```
⍞:
      0 3⊥
          v
        12⊥3 2 8
CORRECT
MORE PROBLEMS?
NO
```

THAT'S IT FOR DECODE!

ENCODE

THE DECODE FUNCTION ENABLES US TO:

* EVALUATE POLYNOMIALS
* DETERMINE DECIMAL EQUIVALENTS OF NUMBERS IN OTHER BASES
* DETERMINE THE COMMON UNIT OF A MIXED UNIT SET

NOW WE MUST LOOK AT THE OTHER SIDE OF THE COIN!

FOR EXAMPLE:

- WHAT IS DECIMAL 27 IN BASE 5?
- HOW MANY YARDS, FEET, AND INCHES IN 3723 INCHES?

THE ENCODE FUNCTION, WHOSE SYMBOL IS ⊤ (UPPER CASE N), PROVIDES US WITH THIS FACILITY.

THUS:

```
      5 5 5⊤27
1 0 2
```

WHICH IS THE BASE 5 *REPRESENTATION OF DECIMAL* 27.

SIMILARLY:

```
      0 3 12⊤3723
103 1 3
```

WHICH TELLS US THAT THERE ARE 103 *YARDS*, 1 *FOOT*, *AND* 3 *INCHES IN* 3723 *INCHES.*

LET'S EXAMINE THE FUNCTION IN MORE DETAIL.

ENCODE IS DYADIC; *ITS GENERAL FORM IS:*

```
  ⍺   A⊤B
```

WHERE: *A* *IS THE BASE INFORMATION*
AND: *B* *IS THE DECIMAL VALUE TO BE CONVERTED*

LET'S DO SOME EXPLORING!
ENTER:
```
      8⊤83
⎕:
      8⊤83
3
```

HMMM! *THAT DOESN'T LOOK RIGHT, DOES IT?*

WELL, TRY:
```
      8 8⊤83
⎕:
      8 8⊤83
2 3
```

NOR DOES THAT!
BUT - YOU'RE MOVING IN THE RIGHT DIRECTION!

NOW ENTER:
```
      8 8 8⊤83
⎕:
      8 8 8⊤83
1 2 3
```

AHA!
NOW WE HAVE THE BASE 8 *REPRESENTATION OF THE DECIMAL NUMBER* 83 (*AS A VECTOR*).

THE POINT IS THAT ENCODING A NUMBER INTO A DIFFERENT BASE REQUIRES 3 *ATA:*

1. *THE DECIMAL VALUE TO BE CONVERTED.*
2. *THE DESIRED NEW BASE.*
3. *THE NUMBER OF DIGIT POSITIONS IN THE CONVERTED NUMBER.*

THUS, GIVEN THAT:

```
      R←A⊤B
```

THEN: $(\rho R)=\rho A$

TO ILLUSTRATE:

THE BINARY (BASE 2) REPRESENTATION OF DECIMAL 13 IS OBTAINED BY:

```
      2 2 2 2⊤13
1 1 0 1
```

BUT THIS IS A VECTOR.

TO OBTAIN THIS VALUE AS A SCALAR, ENTER:

```
      10⊥2 2 2 2⊤13
```
(NOPE! I'M NOT GOING TO DISPLAY THE RESULT!)

YOU TRY IT OUT FOR YOURSELF.
WHEN YOU'VE FINISHED, TYPE: `CONT15`

```
      CONT15
```

LET'S LOOK AT HOW APL EXECUTES THE ENCODE FUNCTION.

ASSUME THAT WE WISH TO KNOW HOW MANY HOURS, MINUTES, AND SECONDS THERE ARE IN 3723 SECONDS. WE WOULD ENTER:

```
      24 60 60⊤3723
1  2  3
```

MEANING 1 HOUR, 2 MINUTES, AND 3 SECONDS.

THIS WAS EVALUATED LIKE THIS:

```
      24 60 60⊤3723
      |  |  |   |
      |  |  |   |
         60|3723   A 3, THE RIGHTMOST COMPONENT OF R
      |  |
      |  |__60|62     A 2, THE NEXT COMPONENT OF R
      |   0          62 IS ⌊3723÷60
      |_____24|1      A 1, THE LAST COMPONENT OF R
                     1 IS ⌊62÷60
```

THIS COMPLETES LESSON 17.

THE NEXT MODULE IS "46 LESSON18" WHICH DISCUSSES THE MONADIC AND DYADIC USE OF THESE FUNCTIONS:

⍒ ⍋ ⌽ ⍉ ⊖ ⍎

)LOAD 46 LESSON18
SAVED 18.33.25 10/06/75
)WIDTH 70
WAS 120
START

LESSON 18

IN THIS LESSON WE SHALL LOOK AT 9 MORE PRIMITIVE FUNCTIONS.
THEY ARE:

MONADIC: ⌽ ⊖ ⍉ ⍋ ⍒
DYADIC: ⍵⌽ ⍺⊖ ⍺⍉ ⍺⍋ ⍺⍒

DO YOU WISH TO SKIP ANY?

NO

IMPORTANT!

WHENEVER I ASK YOU TO SPECIFY VARIABLES OF YOUR OWN CHOOSING IN THIS LESSON, PLEASE USE NAMES WHICH MEET THESE SPECIFICATIONS:

1. ALPHABETIC
2. UNDERSCORED
3. ≥3 CHARACTERS

THIS WILL AVOID ACCIDENTAL MODIFICATION TO GLOBAL VARIABLES USED BY THE FUNCTIONS.

THANK YOU.

REVERSE

OFTEN ONE FINDS IT CONVENIENT TO TURN THINGS AROUND, TO REVERSE THE ORDER IN WHICH ITEMS APPEAR. THE REVERSE FUNCTION, ⌽, (UPPERCASE O,BACKSPACE, UPPERCASE M) PERFORMS THIS FUNCTION. IT'S SYNTAX IS

```
      ⌽A
```

WHERE A MAY BE ANY RANK ARRAY.

I'VE ALREADY DEFINED A VECTOR, W, WOULD YOU PLEASE:
1. DISPLAY W
2. ENTER:⌽W (SEE IF YOU DO INDEED GET A REVERSAL).

THEN TRY THIS FUNCTION ON SEVERAL OTHER VECTORS, INCLUDE A CHARACTER VECTOR, TOO. WHEN YOU'VE DONE THAT, TYPE: MR1.

```
      W
12 24 38 45 52
      ⌽W
52 45 38 24 12
```

```
      ⍳6
1 2 3 4 5 6
      ⌽⍳6
6 5 4 3 2 1
      'ROMMEL'
ROMMEL
      ⌽'ROMMEL'
LEMMOR
      MR1
```

THERE IS NOTHING MORE TO DISCUSS ABOUT VECTOR REVERSAL.
*BUT - I SAID REVERSAL COULD BE APPLIED TO **ANY** RANK ARRAY!*
SO, ADDITIONAL INFORMATION WILL BE NEEDED.

WITH ARRAYS OF RANK ≥ 2 WE MAY SELECT THE DIMENSION ALONG
WHICH REVERSAL IS TO OCCUR, AND DO IT BY INDEXING THE FUNCTION:

⌽[*I*]*A* (*WHERE "I" INDICATES THE DESIRED DIMENSION*)

I HAVE AN ARRAY ALREADY DEFINED IN THIS WS, CALLED: AM
I WOULD LIKE YOU TO:

1. *DISPLAY IT*
2. *DETERMINE ITS SHAPE*
3. *EXECUTE THESE 3 EXPRESSIONS AND COMPARE THE RESULTS WITH "AM":*
   ⌽[1]*AM*
   ⌽[2]*AM*
   ⌽*AM*
4. *DEFINE A (SMALL) MATRIX OF YOUR OWN, AND TRY REVERSING IT IN BOTH DIRECTIONS.*

WHEN YOU HAVE COMPLETED THAT, TYPE: MR2

```
      AM
HOW
IS
THAT
      ⍴AM
3 4
      ⌽[1]AM
THAT
IS
HOW
      ⌽[2]AM
 WOH
  SI
TAHT
      ⌽AM
 WOH
  SI
TAHT
      2 3⍴⍳6
 1  2  3
 4  5  6
      ⌽[1]2 3⍴⍳6
 4  5  6
 1  2  3
      ⌽[2]2 3⍴⍳6
 3  2  1
 6  5  4
      ⌽2 3⍴⍳6
3
      ⌽2 3⍴⍳6
3
```

```
      ϕ(2 3⍴⍳6)
3  2  1
6  5  4
```

MR2

AS YOU HAVE SEEN FROM THE PREVIOUS ILLUSTRATION

ϕ[1]*AM* *PRODUCES A REVERSAL ALONG THE FIRST DIMENSION RESULTING IN A REVERSAL BY ROWS.*

ϕ[2]*AM* *PRODUCES A REVERSAL ALONG THE SECOND DIMENSION, RESULTING IN A REVERSAL BY COLUMNS.*

ϕ*AM* *ALSO PRODUCES A REVERSAL ALONG THE SECOND DIMENSION.*

NOTE: IF YOU DO NOT EXPLICITLY SPECIFY THE DIMENSION ALONG WHICH REVERSAL IS TO OCCUR, THE DEFAULT CONDITION IS THE RIGHT-MOST.

NOW YOU DEFINE A RANK 3 ARRAY AND DO A REVERSAL IN EACH OF ITS DIMENSIONS.

WHEN YOU ARE READY TO CONTINUE TYPE: MR3

```
      PEPE←2 3 4⍴⍳12
      PEPE
 1   2   3   4
 5   6   7   8
 9  10  11  12

 1   2   3   4
 5   6   7   8
 9  10  11  12
     ϕ[1]PEPE
 1   2   3   4
 5   6   7   8
 9  10  11  12

 1   2   3   4
 5   6   7   8
 9  10  11  12
      PEPE←2 3 4⍴⍳50
      PEPE
 1   2   3   4
 5   6   7   8
 9  10  11  12

13  14  15  16
17  18  19  20
21  22  23  24
     ϕ[1]PEPE
13  14  15  16
17  18  19  20
21  22  23  24

 1   2   3   4
 5   6   7   8
 9  10  11  12
     ϕ[2[
        v
        ]PEPE
 9  10  11  12
 5   6   7   8
```

```
 1
      2    3    4

 21   22   23   24
 17   18   19   20
 13   14   15   16
       ⌽[3]PEPE
  4    3    2    1
  8    7    6    5
 12   11   10    9

 16   15   14   13
 20   19   18   17
 24   23   22   21
       MR3
```

THAT'S ALL FOR REVERSE, ⌽[I]A

MONADIC TRANSPOSE

YOU MAY HAVE HEARD ALL KIND OF TALES ABOUT HOW POWERFUL THE TRANSPOSE FUNCTION IS, AND HOW DIFFICULT IT IS TO UNDERSTAND.

LET ME ASSURE YOU THAT THIS IS FAR FROM BEING THE CASE!

THE PRIMITIVE FUNCTION TRANSPOSE: ⍉ (FORMED BY "o", BACKSPACE, "\") WHEN USED MONADICALLY DOES ONLY THIS: ~~IT RESHAPES ITS ARGUMENT SUCH THAT THE RIGHT-MOST 2 DIMENSIONS ARE INTERCHANGED.~~ ~~IT'S THAT SIMPLE!~~

TRANSPOSE REVERSES THE ORDER OF ALL (!) DIMENSIONS.

YOU CAN VERIFY THIS FOR YOURSELF QUITE EASILY - I'VE ALREADY DEFINED AN ARRAY, ZZ, AND IF YOU:

1. *DETERMINE ITS SHAPE*
2. *DISPLAY IT*
3. *TRANSPOSE IT, AND COMPARE THE RESULT TO THE ORIGINAL*

YOU'LL SEE WHAT I MEAN.

EXECUTE THESE STEPS NOW, AND, WHEN YOU'RE THROUGH, TYPE: MT2

```
      ZZ
 1    2    3    4
 5    6    7    8
 9   10   11   12

13   14   15   16
17   18   19   20
21   22   23   24
      ⍴ZZ
2 3 4
      ⍉ZZ
 1   13
 5   17
 9   21

 2   14
 6   18
10   22

 3   15
 7   19
```

(2 3 4 without 4 3 2)

```
 1 1  23

  4  16
  8  20
 12  24
      ρ⍉ZZ
4 3 2
      ALFA←2 3 4 5ρ⍳100
      ρALFA
2 3 4 5
      ρ⍉ALFA
5 4 3 2
      ALFA←2 3ρ⍳6
      ρALAF
          v
          FA
2 3
      ρ⍉ALFA
3 2
      ALFA←2 3 4 5 6ρ⍳12
      ρALFA
2 3 4 5 6
      ρ⍉ALFA
6 5 4 3 2
      MT2
```

OK.
NOW TRY MONADIC TRANSPOSE ON A MATRIX AND A VECTOR (THAT'S RIGHT - A VECTOR!) AND TO MAKE LIFE EASIER FOR YOU, I'VE ALREADY DEFINED A MATRIX, AM, AND A VECTOR, W, FOR YOU TO USE.

WHEN YOU'VE DONE THAT, TYPE: MT3

```
      AM
HOW
IS
THAT
      ⍉AM
HIT
OSH
W A
  T
      W
12 24 38 45 52
      ⍉W
12 24 38 45 52
      MT3
```

YOU WILL HAVE NOTICED THAT, IN THE CASE OF A VECTOR, TRANSPOSE HAD NO EFFECT (WHICH IS FAR BETTER THAN AN ERROR MESSAGE!), BUT THAT IT WORKED WELL ON RANK 2 ARRAYS.

THUS, MONADIC TRANSPOSE (⍉) INTERCHANGES THE LAST 2 DIMENSIONS OF A RANK ≥ 2 ARRAY.

AND THAT'S ALL THERE IS TO MONADIC TRANSPOSE (THERE IS, THOUGH, MUCH MORE ABOUT DYADIC TRANSPOSE.)

MONADIC REFLECT

THE SYMBOL FOR THE REFLECT FUNCTION IS: ⌽ (⍥ *OVERSTRUCK BY* ⍤)

AND IT IS NOTHING MORE THAN: ⌽[1] (FIRST DIMENSION ROTATE).

TO FIND OUT HOW IT WORKS, DO THIS:

1. DISPLAY: AM ZZ V
2. DETERMINE THEIR SHAPES
3. EXECUTE: ⊖AM ⊖ZZ ⊖V
4. COMPARE THE RESULTS TO THE ORIGINALS

WHEN YOU'VE DONE THAT, TYPE: MF2

```
      AM
HOW
IS
THAT
      ρAM
3 4
      ⊖AM
THAT
IS
HOW
      ρ⊖AM
3 4
      MF2
```

AND THAT'S ALL THERE IS TO SAY ABOUT MONADIC REFLECT: ⊖

GRADE UP

THE GRADE UP FUNCTION, ⍋ (UPPER CASE "H" OVERSTRUCK BY UPPER CASE "M"), IS ALMOST SIMILAR TO A "SORT" IN ASCENDING SEQUENCE. ALMOST - BUT NOT QUITE!

WHAT IT ACTUALLY DOES IS YIELD THE INDICES OF THE COMPONENTS OF THE VARIABLE IN ASCENDING SEQUENCE.

I'VE ALREADY DEFINED A VECTOR, A, AND I'D LIKE YOU TO DO THIS:

1. DISPLAY A
2. ENTER: ⍋A (AND EXAMINE THE RESULTS)

WHEN YOU'VE DONE THAT, TYPE: GCON1

```
      A
80 45 62 37 29 74 58 15 96
      ⍋A
8 5 4 2 7 3 6 1 9
      GCON1
```

YOU WILL HAVE NOTICED THAT THE RESULT OF ⍋A WAS NOTHING LIKE THE DATA IN "A" ITSELF!
(AS I SAID, IT YIELDS THE INDICES OF THE DATA IN "A".)

WELL, SUPPOSING WE WISHED TO ACTUALLY SORT "A" SO THAT ITS COMPONENTS WERE
IN ASCENDING SEQUENCE (BASED UPON THE NUMERICAL VALUE FOR EACH)?

NO PROBLEM!

ALL WE'D DO IS TO ENTER:

```
      A[⍋A]
```

AND WE'D GET.....OK! YOU TRY IT! SEE WHAT YOU GET.

WHEN YOU'VE DONE THAT, TYPE: GCON2

```
      A[⍋A]
15 29 37 45 58 62 74 80 96
      GCON2
```

AND ALL WE WOULD HAVE TO DO TO REPLACE "A" WITH A SORTED VERSION OF "A" IS:

```
      A←A[⍋A]
```

WHICH IS ABOUT ALL WE CAN SAY FOR GRADE UP.

GRADE DOWN

THE GRADE DOWN FUNCTION, ⍒ (UPPER CASE "G" OVERSTRUCK BY UPPER CASE "V"), IS LIKE A SORT IN DESCENDING SEQUENCE - EXCEPT THAT INSTEAD OF SORTING THE COMPONENTS, IT YIELDS THEIR INDICES IN THE SEQUENCE OF THEIR DESCENDING NUMERICAL VALUES.

THERE IS A GLOBAL VARIABLE, A, WHICH IS A CONSTANT VECTOR. DISPLAY IT, AND THEN ENTER: ⍒A

EXAMINE THE OUTPUT, AND, WHEN YOU HAVE DONE SO, TYPE: GCON3

```
      A
80 45 62 37 29 74 58 15 96
      ⍒A
9 1 6 3 7 2 4 5 8
      A[∇
⌈↑T~⍺⊃ ∈⍴⍴O⍴
      A[
        ^
      A[⍒A]
96 80 74 62 58 45 37 29 15
      GCON3
```

JUST AS IN GRADE UP, ONLY THE INDICES WERE SORTED - NOT THE COMPONENTS!

AND, OF COURSE, JUST AS FOR GRADE UP, WE CAN OBTAIN THE COMPONENTS IN DESCENDING ORDER BY ENTERING:

```
      A[⍒A]
```

TRY IT AND SEE.

WHEN YOU'RE SATISFIED, TYPE: GCON4

```
      GCON4
```

THAT'S REALLY ALL YOU NEED TO KNOW ABOUT GRADE DOWN!

ROTATION

THE MONADIC FUNCTION REVERSAL, ⌽A, PRODUCES A COMPLETE REVERSAL; HOWEVER, WE OFTEN WANT ONLY A PARTIAL REVERSAL. THE DYADIC FUNCTION

ROTATE

A⌽B

GIVES US THIS FACILITY. THE ROTATE FUNCTION ALLOWS US TO ROTATE ARRA
DATA IN A CIRCULAR FASHION.

LET'S SEE HOW IT WORKS. I 'VE DEFINED A VECTOR V, PLEASE DO THE FOLL
WING:

1. DISPLAY V
2. ENTER: 2⌽V (COMPARE THE RESULT WITH V)
3. ENTER: ¯1⌽V (COMPARE THE RESULT WITH V)
4. ENTER: 8⌽V (COMPARE THE RESULT WITH V)
5. ENTER: 0⌽V (COMPARE THE RESULT WITH V)
6. ENTER .5⌽V (DON'T BE SURPRISED)

WHEN YOU HAVE FINISHED TYPE:)DR1

```
      V
2 7 4 9 10 23 45 52 12 26 3 8 5
      2⌽V
4 9 10 23 45 52 12 26 3 8 5 2 7
      ¯1⌽V
5 2 7 4 9 10 23 45 52 12 26 3 8
      8⌽V
12 26 3 8 5 2 7 4 9 10 23 45 52

      0⌽V
2 7 4 9 10 23 45 52 12 26 3 8 5
      .5⌽V
DOMAIN ERROR
      0.5⌽V
      ^
      DR1
```

YOUR EXERCISES SHOULD HAVE INDICATED TO YOU THAT THE LEFT ARGUMENT
OF DYADIC ROTATE DETERMINES THE DEGREE OF ROTATION, WITH A POSITIVE
VALUE RESULTING IN ROTATION TO THE LEFT, AND A NEGATIVE LEFT ARGUMENT
RESULTING IN ROTATION TO THE RIGHT.

THUS:

```
    2⌽V    ROTATES "V" 2 PLACES TO THE LEFT
   ¯1⌽V    ROTATES "V" 1 PLACE TO THE RIGHT
```

ALSO, YOU WILL RECALL THAT THE EXPRESSION:

```
    .5⌽V
```

PRODUCED A "DOMAIN ERROR" MESSAGE.
THIS INDICATES THAT THE LEFT ARGUMENT MUST ALSO BE AN INTEGER.

NOW, LET'S RETURN TO OUR MATRIX, "AM".
WITH ARRAYS OF RANK ≥ 2 WE CAN SELECT THE DIMENSION ALONG
WHICH ROTATION IS TO OCCUR, WHICH YIELDS THIS GENERAL FORM
FOR THE ROTATE FUNCTION:

```
    A⌽[I]B    (WHERE "I" INDICATES THE DESIRED DIMENSION)
```

BUT THAT'S NOT ALL! THERE IS AN ADDITIONAL FEATURE AVAILABLE.
TRY TO DETERMINE WHAT IT IS FROM THE FOLLOWING EXERCISE SET:

1. DISPLAY ⍴AM
2. DISPLAY AM

3. *EXECUTE EACH OF THE FOLLOWING EXPRESSIONS, AND COMPARE THE RESULT WITH "AM":*

```
      0 1 2⌽AM
      0 ¯1 2⌽[2]AM
      1 ¯1 1⌽AM
      1 0 ¯1 0⌽[1]AM
      ¯1 ¯2 1 2⌽[1]AM
```

WHEN YOU'VE FINSIHED ALL THAT, TYPE: DR2

```
      ⍴AN
        ∨
        M
3 4
      AM
HOW
IS
THAT
      0 1 2⌽AM
HOW
S  I
ATTH
      DR2
```

FROM THE ABOVE EXERCISES YOU SHOULD HAVE OBSERVED THAT NOT ONLY CAN YOU CHOOSE THE DIMENSION IN WHICH ROTATION IS TO OCCUR BUT ALSO YOU MAY ROTATE IN EACH OF THE OTHER DIMENSIONS INDEPENDENTLY. RECALL IN

```
      0 1 2⌽AM  (OR  0 1 2⌽[2] AM)
```

ROTATION WAS ALONG THE SECOND DIMENSION. FOR AM, THIS IS A ROTATION BY ROW, WHERE THE AMOUNT OF ROTATION FOR EACH ROW IS INDEPENDENT OF THAT FOR THE OTHERS. THIS LEADS TO THE FOLLOWING GENERAL RULES FOR

```
      A⌽[I]B
```

1. $(\rho\rho A) = {}^{-}1+\rho\rho B$ — dimensie van A is een lager dan van B.
2. $(\rho A) = (I \neq \iota \rho B)/\rho B$ aantal elementen van A moet gelijk zijn aan het aantal elementen van de gekozen dimensie van B.

HERE'S ANOTHER EXERCISE SET:
I'VE ALREADY DEFINED THESE VARIABLES -

AR3 RR BR PR

AND YOU ARE TO:

1. *DETERMINE THEIR SHAPES*
2. *DISPLAY THEM*
3. *EXECUTE THE FOLLOWING (WHICH MAY OR MAY NOT WORK....):*

```
      RR⌽AR3
      BR⌽[2]AR3
      PR⌽[1]AR3
      BR⌽[1]AR3
```

AND COMPARE THE RESULTS WITH "AR3".

WHEN YOU HAVE COMPLETED THIS, TYPE: DR3

```
      AR3
ABC
123
DEF
9- 456

GHI
789
```

⍴AR3: 2 4 3

```
JKL
012

      RR
 1  0  2  0
 1  1  0  2
      BR
 0  1  2
 2  1  0
      PR
0 1 0
1 0 1
0 1 0
1 0 1
      RR⌽AR3
BCA
123
FDE
456

HIG
897
JKL
201
      BR⌽AR3
INDEX ERROR
      BR⌽AR3
      ^
      PR⌽AR3
ABC
231
DEF
564

GHI
897
JKL
120
      DR3
THIS COMPLETES OUR PRESENTATION ON ROTATE,  A⌽[I]B.
```

DYADIC TRANSPOSE

THE EXPLANATIONS OF THE WAY DYADIC TRANSPOSE OPERATES HAVE, HISTORICALLY, BEEN UNNECESSARILY COMPLICATED.

INSTEAD OF GOING INTO A LENGTHY, RIGOROUS DEFINITION OF WHAT HAPPENS, WE'LL TRY THIS:

1. I HAVE AN ARRAY, AR3, IN THIS WS - DISPLAY IT
2. DETERMINE ITS SHAPE
3. EXECUTE: 3 2 1⍉AR3 AND COMPARE THE RESULT WITH AR3

WHEN YOU'VE DONE THAT, TYPE: DT2

```
      AR3
ABC
123
DEF
456

GHI
789
JKL
012
```

```
2 4 3 ⍴AR3
      3 2 1⍉AR3
AG
17
DJ
40

BH
28
EK
51

CI
39
FL
62
      DT2
```

WHAT HAPPENED WAS THAT THE LEFT ARGUMENT OF DYADIC TRANSPOSE ACTED LIKE AN INDEX, BY POSITION, ON THE DIMENSIONS OF THE RIGHT ARGUMENT, AND APL EXECUTED YOUR EXPRESSION LIKE THIS:

```
                                        R←3 2 1 ⍉ AR3
                                          |  |  |
READS AS FOLLOWS:                         |  |  |
   THE FIRST DIMENSION OF  AR3            |  |  |
       BECOMES                    -------'  |  |
     THE THIRD DIMENSION OF R               |  |
                                            |  |
   THE SECOND DIMENSION OF AR3              |  |
       BECOMES                    ---------'  |
     THE SECOND DIMENSION OF R                |
                                              |
   THE THIRD DIMENSION OF AR3                 |
       BECOMES              0     -----------'
     THE FIRST DIMENSION OF R
```

THERE IS A GREAT DEAL MORE THAT I COULD SAY ABOUT THE TRANSPOSE FUNCTION, BUT I THINK IT WOULD ONLY SERVE TO CONFUSE, AND, IN ANY CASE, YOU NOW HAVE ENOUGH UNDERSTANDING OF ITS OPERATION TO EITHER EXPERIMENT, OR TO USE IT.

DYADIC REFLECT

DYADIC REFLECT IS NOTHING MORE THAN ROTATION ALONG THE FIRST DIMENSION. ITS SYMBOL IS ⊖. LET'S LOOK AT IT.

DO THIS:

1. *DISPLAY AM ZZ RM*
2. *DETERMINE THEIR SHAPES*
3. *EXECUTE:*

```
      0 1 2 ¯2⊖AM
      RM⊖ZZ
```

4. *COMPARE THE RESULTS TO THE ORIGINALS.*

WHEN YOU'VE FINSHED THAT, TYPE: DR4

```
      AM
```

HOW
IS
THAT

1-

```
      0 1 2 ¯2⊖AM
HSA
IHWT
TO
      DR4
```

THAT'S ALL I HAVE TO SAY, EXCEPT TO POINT OUT THAT THE EXPRESSION:

```
     ¯1⊖A
```

IS ONE OF THE MOST USEFUL YOU'LL COME ACROSS FOR MAINTAINING THE "N" (WHERE N←(⍴A)[1]) MOST RECENT SETS OF DATA.

COMPRESS ALONG 1ST DIMENSION

YOU ARE ALREADY FAMILIAR WITH THE COMPRESS FUNCTION, '/', BUT WE HAVE NOT YET DISCUSSED USING IT ON ARRAYS.

ESSENTIALLY, THE RULE IS THIS:

COMPRESS OPERATES ON THE LAST DIMENSION, UNLESS INDEXED

LET ME EXPLAIN THAT.

SUPPOSE THAT: `Y←3 4⍴⍳12`

THEN:

```
    0 1 0 1/Y
 2  4
 6  8
10 12
```

AND:

```
     0 1 0⌿Y
5 6 7 8
```

WHICH IS THE SAME AS: `0 1 0/[1]Y`

NOW YOU CAN TRY IT FOR YOURSELF.

SPECIFY SOME ARRAYS (BUT DON'T USE THE VARIABLE NAME "A") AND SEE WHAT HAPPENS WHEN YOU COMPRESS ALONG DIFFERENT DIMENSIONS.

WHEN YOU'RE SATISFIED, TYPE: COMP2

```
      Y
VALUE ERROR
      Y
      ∧
      ALEA←3 4⍴⍳12
      ALEA
 1  2  3  4
 5  6  7  8
 9 10 11 12
```

```
      0 1 0 1/ALFA
  2   4
  6   8
 10  12
      011/[1]ALFA
DOMAIN ERROR
      11/[1]ALFA
      ^
      0 1 1/[1]ALFA
  5   6   7   8
  9  10  11  12
      COMP2
```

AS YOU SAW, "⌿" IS NOTHING MORE THAN COMPRESS ALONG THE FIRST DIMENSION - OR IS IT?

AS A MATTER OF FACT, IT IS ONLY <u>COMPRESSION</u> WHEN ITS LEFT ARGUMENT IS A LOGICAL VECTOR. OTHERWISE, IF ITS LEFT ARGUMENT IS A PRIMITIVE, SCALAR, DYADIC FUNCTION, IT WILL YIELD <u>REDUCTION</u> ALONG THE FIRST DIMENSION!

FOR EXAMPLE:

```
      [⌊←M←2 3⍴⍳6

 1  2  3
 4  5  6
```

NOW:

```
      +/M
6 15
```

AND:

```
      +⌿M
5 7 9
```

AND THAT'S ALL THERE IS TO DISCUSS!

THIS COMPLETES LESSON 18.

*THE NEXT MODULE IS "*46 *LESSON*19*",*
WHICH DEALS WITH MORE SYSTEM COMMANDS.

)LOAD 46 LESSON19
SAVED 18.35.54 10/06/75
)WIDTH 70
WAS 120
START

IN THIS SECTION WE WILL LOOK AT A NUMBER OF ADDITIONAL SYSTEM COMMANDS THAT WERE NOT COVERED IN MODULE 2 OF THE COURSE. THE SECTION WILL BE ORGANIZED INTO FIVE GROUPS:

1. UTILITY COMMANDS
2. TERMINAL-TO-TERMINAL COMMUNICATION
3. GROUPING
4. THE STATE INDICATOR
5. MISCELLANEOUS

LET'S TAKE A LOOK AT THE 'UTILITY' COMMANDS --

)WIDTH N - SETS THE WIDTH OF THE PRINTABLE PAGE TO N CHARACTERS (30≤N≤130)

)DIGITS N - SETS THE NUMBER OF SIGNIFICANT DIGITS IN A NUMBER PRINTED TO N (1≤N≤16)

)ORIGIN N - SETS THE INDEX ORIGIN FOR THIS WORKSPACE (N= 0 OR 1)

LET'S TRY THEM --

NOW - YOU CHANGE THE PRINTABLE PAGE WIDTH TO 40 CHARACTERS:

)WIDTH 40

CORRECT!

NOW - LET'S PROVE THAT THE WIDTH HAS BEEN CHANGED.

SPECIFY A LITERAL VECTOR, V, OF LENGTH
GREATER THAN 40 CHARACTERS, AND I'LL PRINT IT:

□:
60ρ'WHMJ '
WHMJ WHMJ WHMJ WHMJ WHMJ WHMJ WHMJ WHMJ
WHMJ WHMJ WHMJ WHMJ

GOOD.

NOW I'LL CHANGE THE WIDTH BACK TO 130.
NOW CHANGE THE NUMBER OF PRINTED SIGNIFICANT DIGITS TO 3:

```
)DIGITS 3
WAS 3

VERIFY FOR YOURSELF THAT THE CHANGE HAS BEEN MADE
BY ENTERING AN ARITHMETIC EXPRESSION WHICH
WILL RESULT IN MORE THAN 3 SIGNIFICANT DIGITS.

TYPE: CONTINUE
TO PROCEED.
⎕:
      2*.5
1.41
⎕:
      CONTINUE

OK.
NOW I'LL CHANGE "DIGITS" BACK TO 10.

WAS 3


NOW LET'S TAKE A LOOK AT THE "ORIGIN" COMMAND.
WE ARE PRESENTLY USING AN INDEX ORIGIN OF 1.
WHICH CAUSES THE EXPRESSION: ⍳10  TO YIELD:

     ⍳10
1 2 3 4 5 6 7 8 9 10
CHANGE THE INDEX ORIGIN TO ITS OTHER VALUE:

)ORIGIN 0
WAS 0

VERIFY THE CHANGE OF INDEX ORIGIN TO 0
(SAY, BY: ⍳10) AND TYPE: CONTINUE
TO PROCEED.

⎕:
      ⍳10
0 1 2 3 4 5 6 7 8 9
⎕:
      CONTINUE

OK.
NOW I'LL CHANGE IT BACK TO 1.

WAS 1

THOSE ARE THE SO-CALLED "UTILITY COMMANDS".
YOU WILL FIND THEM VERY USEFUL.
```

TERMINAL-TO-TERMINAL COMMUNICATION

FROM TIME TO TIME YOU MAY FIND IT NECESSARY TO COMMUNICATE WITH EITHER THE APL OPERATOR, OR WITH SOMEONE ELSE WHO IS ON THE SYSTEM.

YOU CAN USE THE APL SYSTEM FOR THIS, WITH THESE SYSTEM COMMANDS:

`)OPR |←---MESSAGE---→|` - THIS SENDS THE TEXT OF THE MESSAGE TO THE APL OPERATOR, AND LOCKS YOUR KEYBOARD UNTIL A REPLY IS RECEIVED.

)PORTS - PRINTS A LIST OF ALL PORT NUMBERS IN USE, AND THE 3-CHARACTER KEY OF WHO IS USING EACH.

)MSG X |←--MESSAGE-→| - TRANSMITS THE TEXT OF THE MESSAGE TO THE TERMINAL ON PORT "X", AND, AGAIN, LOCKS YOUR KEYBOARD UNTIL A REPLY IS RECEIVED.

IF YOU WISH TO SEND A MESSAGE, BUT DO NOT WANT A REPLY, THEN USE THE COMMANDS:

)OPRN |←----MESSAGE----→| - SAME AS ")OPR", BUT NO KEYBOARD LOCK

)MSGN X |←---MESSAGE---→| - SAME AS ")MSG X", BUT NO KEYBOARD LOCK.

NOW LET'S TRY THEM OUT.....

SEND A MESSAGE TO THE APL OPERATOR, AND ASK IF THE SYSTEM WILL BE "UP" TONIGHT - WAIT FOR A REPLY:

```
)OPR WILL THE SYSTEM BE "UP" TONIGHT?
SENT
OPR: NO, IT WILL BE DOWN FOR MAINTENANCE TONIGHT
```

GOOD.

HERE'S A PROBLEM FOR YOU:

YOU HAVE A FRIEND NAMED A.E.NEUMAN.
FIND OUT IF HE'S ON THE SYSTEM, AND,
IF HE IS, SEND HIM A MESSAGE.

(PLEASE USE ONLY LETTERS OR DIGITS IN YOUR MESSAGE TEXT.)

```
)PORTS
  02 WHN
  04 RAJ
 006 RHO
  07 PGA
  08 AEN
  10 HHM
  31 OPE
```

OK SO FAR!
NOW YOU KNOW HE'S ON THE SYSTEM.
KEEP GOING.....

```
)MSG 08 SYSTEM MESSAGE
SENT
008: I'LL CALL YOU LATER.      .....ALF
```

VERY GOOD!

THE GROUPING COMMANDS CAN BE SUMMARIZED AS FOLLOWS:

)GROUP GPNAME OBJ1 OBJ2.....OBJ(N) - COLLECTS THE SPECIFIED OB

ECTS

.

MOVED

G THE

INTO A GROUP CALLED GPNAM-
THESE OBJECTS CAN THEN BE
COLLECTIVELY BY REFERENCIN
GROUP NAME.

)GROUP GPNAME - WILL ERASE THE GROUP CALLE
D GPNAME

FROM THE ACTIVE WORKSPACE

- ONLY

THE GROUP DEFINITION - NO
OBJECTS THEMSELVES! (YOU C
A GROUP BY EXECUTING:)ERA

THE

AN ERASE

ERASE GPNAME)

)GRPS - LISTS THE NAMES OF ALL GRO
UPS IN

THE ACTIVE WORKSPACE.

)GRP GPNAME - LISTS ALL THE OBJECTS IN T-
HE GROUP

CALLED GPNAME.

)GROUP GPNAME GPNAME OBJ7 OBJ8.... - ADDS THE OBJECTS TO THE AL
READY

EXISTING GROUP CALLED GPN.

ME.

THIS IS PRETTY SIMPLE, SO LET'S SEE HOW YOU DO.....

LIST THE GROUPS IN THIS WS:

)GRPS

DOGS CATS BIRDS

GOOD!

NOW - LIST THE MEMBERS OF ONE OF THE GROUPS:

)GRP DOGS

SETTER SPANIEL BOXER

TRY THIS ONE AGAIN?
NO

OK.

DEFINE A NEW GROUP OF YOUR OWN:

)GROUP RATJETOE A1 A2

GOOD SHOW!

WE SHALL NOW DISCUSS TWO COMMANDS WHICH REFER TO THE STATE INDICATOR:

)SI - LIST ALL "HALTED" FUNCTIONS IN THE ACTIVE WS, WITH THE MOST RECENTLY HALTED FUNCTION AT THE TOP OF THE LIST. FUNCTIONS MARKED WITH AN ASTERISK (*) ARE CALLED "SUSPENDED" FUNCTIONS; THOSE WHICH HAVE NO * ARE CALLED "PENDENT" FUNCTIONS. LINES ABOUT TO BE EXECUTED WHEN THE INTERRUPT OCCURRED ARE BRACKETED FOLLOWING EACH HALTED FUNCTION NAME.

)SIV - DISPLAYS THE STATE INDICATOR AS IN ")SI", BUT PRINTS ALONGSIDE EACH HALTED FUNCTION THOSE LOCAL VARIABLES ASSOCIATED WITH IT.

WE'D BETTER DEFINE SOME OF THOSE TERMS.....

A FUNCTION IS CALLED:	IF THESE CONDITIONS EXIST:
SUSPENDED	1) IT IS WAITING TO BE RE-STARTED
	2) IT IS DEFINED (AS OPPOSED TO PRIMITIVE)
	3) IT CAN BE EDITED (BECAUSE IT IS SUSPENDED)
PENDENT	1) IT CALLS ANOTHER FUNCTION (PROGRAM)
	2) IT IS DEFINED
	3) IT IS WAITING TO BE RESUMED

NOTE: A PENDENT FUNCTION CANNOT BE EDITED - APL WILL SEND YOU A "DEF ERROR" MESSAGE
IF YOU TRY! SEE IF YOU CAN FIGURE OUT WHY THIS SHOULD BE SO
(BUT DO IT LATER BECAUSE NOW WE'RE GOING TO TRY A FEW PROBLEMS
)

HERE'S A LIST OF WORDS OR PHRASES
WHICH DESCRIBE FUNCTIONS:

1 - PRIMITIVE
2 - EDITABLE
3 - WAITING
4 - CALLS ANOTHER PROGRAM (FUNCTION)
5 - DEFINED

ENTER A NUMERIC VECTOR FROM THE ABOVE LIST
WHICH ACCURATELY DESCRIBES A SUSPENDED FUNCTION:

```
⍞:
      1
      ∨

      ∨
      3 5 2
```

RIGHT!

NOW ENTER A VECTOR WHICH ACCURATELY
DESCRIBES A PENDENT FUNCTION:

```
⍞:
      4 5 3
```

VERY GOOD.
NOW DETERMINE IF ANY FUNCTIONS ARE HALTED:

```
)SI
CAT[3] *
DOG[8]
RAT[6] *
BAT[7] *
```

NOW DETERMINE WHICH VARIABLE(S) IS/ARE LOCAL TO FUNCTION "RAT":

```
)⎕LU
 V
 ⎕LU
 V
 SIV
CAT[3] * A B
DOG[8]
RAT[6] * I
BAT[7] * I J
```

PLEASE CONTINUE.
CONTINUE
NO. TRY AGAIN.
I

VERY GOOD.

LET'S TAKE A LOOK AT A COUPLE OF COMMANDS WHICH FALL INTO WHAT I CALL THE "MISCELLANEOUS" CATEGORY:

)PCOPY - THIS WORKS JUST LIKE THE ")COPY" WITH ONE EXCEPT
ON:

IF THERE IS AN OBJECT IN THE ACTIVE WORKSPACE WI
H

AN IDENTICAL NAME AS (ONE OF) THE OBJECT(S) BEIN
COPIED, THE OBJECT ALREADY IN THE ACTIVE WS WILL
REMAIN. THIS IS CALLED A "PROTECT COPY" AND YOU
WILL FIND IT USEFUL WHEN UPDATING A SET OF
RELATED FUNCTIONS.

)CONTINUE - THIS COMMAND DOES THE FOLLOWING:
1. SAVES YOUR ACTIVE WS IN "CONTINUE".
2. SIGNS YOU OFF THE SYSTEM.

YOU CAN PRACTICE USING THESE SOME OTHER TIME.

TWO LAST TOPICS, AND THEN WE'RE THROUGH!

1. LOCKED FUNCTIONS - YOUR FUNCTIONS CAN BE "LOCKED" BY ENTERI-
NG THE
CLOSING DEL (∇) AS ∇ OVERSTRUCK BY ~
WHICH
FOREVER PREVENTS FUTURE DISPLAY OR EDITI
G!
NOBODY (INCLUDING THE APL OPERATOR) CAN
GET AT THEM!
(YOU CAN ERASE AND REPLACE, HOWEVER.)

2.

IBEAM FUNCTIONS - THESE ARE SPECIALIZED SYSTEM FUNCTIONS WHICH ARE LISTED ON PAGE 3.47 OF THE USER'S MANUAL EXAMPLES OF THEIR USE WILL BE FOUND IN LESSON 21.

AND THAT COMPLETES THIS MODULE!

THE NEXT MODULE IS "46 LESSON20", WHICH DISCUSSES INNER AND OUTER PRODUCTS.

```
      )LOAD 46 LESSON20
SAVED  18.39.06 10/06/75
      )WIDTH 70
WAS 120
      SR
       V
       TART
```


LESSON 20

IN THIS MODULE WE SHALL DISCUSS TWO POWERFUL FUNCTIONS UNIQUE TO APL - THE INNER AND OUTER PRODUCTS, AND WE SHALL START WITH THE LATTER.

IN ORDER TO UNDERSTAND OUTER PRODUCTS, IT IS NECESSARY TO HAVE A SOUND KNOWLEDGE OF VECTORS, SO LET'S BEGIN WITH A FEW REVIEW PROBLEMS. IF YOU GET INTO ANY KIND OF DIFFICULTY IN THIS MODULE, JUST TYPE: HELP

PROBLEM 1:
ENTER AN EXPRESSION WHICH WILL DETERMINE THE 3RD POWER OF THE VECTOR 4 5 9

```
⍞:
      4 5 9*3
```

VERY GOOD!

PROBLEM 2:
ENTER AN EXPRESSION WHICH WILL DETERMINE THE 1ST, 2ND, AND 3RD POWERS OF 3

```
⍞:
      3*1 2 3
```

CORRECT!

PROBLEM 3:
SPECIFY "T" TO BE THE 1ST, 2ND, AND 3RD POWERS OF 4 5 9 (RESPECTIVELY) - THAT IS, I WANT A 3-COMPONENT VECTOR AS THE RESULT.

WHEN YOU'VE DONE THAT, TYPE: DONE

```
      T←4 5 9*1 2 3
      DONE
```

GOOD!
WHEN YOU'RE READY TO CONTINUE, TYPE: NEXT

```
      NEXT
```

LET'S CHANGE THE PROBLEM SLIGHTLY:
SPECIFY "T1" AS THE 1ST, 2ND, AND 3RD POWERS OF EACH COMPONENT

OF THE VECTOR 4 5 9 (AND BE CAREFUL!).
⍞:

```
      T1←3 3⍴(4*⍳3),(5*⍳3),(9*⍳3)
```

EXCELLENT!
DO YOU KNOW WHY YOUR ANSWER WAS CORRECT?

NO

THE EXPRESSION:

```
      T1←4 5 9∘.*⍳3
```

GENERATES THIS 3 BY 3 ARRAY:

```
 4   16   64
 5   25  125
 9   81  729
```

WHICH WAS GENERATED AS A TABLE REPRESENTING THE RESULT OF EACH COMPONENT IN THE LEFT ARGUMENT BEING OPERATED UPON, IN TURN, BY EACH COMPONENT OF THE RIGHT ARGUMENT. THE ACTUAL OPERATION WHICH IS PERFORMED DEPENDS UPON THE FUNCTION SPECIFIED.

THE GENERAL FORM OF THE OUTER PRODUCT IS:

```
A∘.(F)B
```

WHERE (F) IS ANY SCALAR, DYADIC FUNCTION.

HERE'S AN OUTER PRODUCT EXPRESSION:

```
(⍳4)∘.+⍳4
```

ENTER THE (F) IN THE ABOVE EXPRESSION:

```
+
```

GOOD!

LET'S EXAMINE HOW APL EVALUATES AN OUTER PRODUCT.

WE SHALL USE THIS EXPRESSION AS OUR EXAMPLE:

```
      (⍳3)∘.×⍳4
```

APL SETS UP A TABLE (MATRIX) WHOSE DIMENSIONS ARE DETERMINED BY THE SHAPES OF THE ARGUMENTS; IN OUR EXAMPLE, THE LEFT ARGUMENT IS A VECTOR OF LENGTH 3, AND THE RIGHT ARGUMENT IS A VECTOR OF LENGTH 4.

APL, THEREFORE, SETS UP A TABLE WITH 3 ROWS AND 4 COLUMNS, AND YOU MIGHT THINK OF IT THIS WAY:

×	1	2	3	4
1				
2				0
3				

AND THE RESULT WILL BE CONTAINED IN THE SQUARES.

EACH SQUARE WILL CONTAIN THE RESULT OF THE ROW VALUE (LEFT ARGUMENT COMPONENT) OPERATED UPON BY ITS COLUMN VALUE (RIGHT ARGUMENT COMPONENT).

SO, LET'S EXECUTE OUR EXAMPLE AND SEE:

```
      1 2 3∘.×⍳4

  1   2   3   4
  2   4   6   8
  3  06   9  12
```

AND NOW IT'S YOUR TURN.....

ENTER AN EXPRESSION WHICH YIELDS THE FOLLOWING:

```
      0                    0  0                 0
      1                    1                    1
      2                 0  4                    8
⍞:
      0 1 2∘.*1 2 3
```

CORRECT!

NOW LET'S EXAMINE THE INNER PRODUCT.

HERE'S AN EXAMPLE:

```
      4 5 9+.×⍳3
41
```

WHICH APL ARRIVED AT THIS WAY:

```
     +/4 5 9×⍳3
```

AND, THAT'S REALLY THE DEFINITION OF INNER PRODUCTS IN A NUTSHELL!

THE GENERAL FORM OF THE INNER PRODUCT IS:

A (F1).(F2) B (*WHERE (F1) AND (F2) ARE ANY SCALAR, DYADIC FUNCTIONS*)

AND THE DEFINITION, IN WORDS, IS:

> *(F2) IS APPLIED IN A SCALAR FASHION TO BOTH ARGUMENTS, AND THE RESULT OF THAT IS (F1) REDUCED.*

ASSUME THAT V1←⍳10 AND V2←1+⍳10 (NOTE THEIR LENGTHS ARE EQUAL!).

THEN:

```
      V1∧.=V2
0
```

NOW: ENTER AN INNER PRODUCT EXPRESSION WHICH DETERMINES IF V2 IS GREATER THAN V1:

```
⍞:
      (V2+.×V2)>(V1+.×V1)
```

GREAT!

HERE'S AN EXAMPLE OF A COMBINED USE FOR AN INNER AND AN OUTER PRODUCT:

FIRST, WE'LL EXECUTE THIS OUTER PRODUCT:

```
      (⍳5)∘.≤⍳5
  | 1 2 3 4 5
--+-----------
1 | 1 1 1 1 1
2 | 0 1 1 1 1
3 | 0 0 1 1 1
4 | 0 0 0 1 1
5 | 0 0 0 0 1
```

AND THIS VECTOR:

```
      ?5⍴100
76  46  54  22  5
```

NOW, LET'S ASSUME THAT WE HAVE A VECTOR (SUCH AS THE RANDOM ONE WE JUST GENERATED) AND WE WISHED TO MAKE EACH COMPONENT THE SUM OF ITSELF AND ALL PRECEDING COMPONENTS, WHICH, IN THE EXAMPLE, WOULD YIELD:

```
76  122  176  198  203
```

WE CAN DO THIS SIMPLY BY EXECUTING:

```
      76 46 54 22 5+.×(⍳5)∘.≤⍳5
76  122  176  198  203
```

AND TO PUT IT INTO A MORE GENERAL FORM:

```
      V←V+.×(⍳⍴V)∘.≤⍳⍴V
```

WHICH IS ENOUGH FOR THAT TOPIC!

HERE'S A LITTLE PROBLEM FOR YOU:

IT FREQUENTLY HAPPENS THAT COMPARISONS BETWEEN VARIABLES OF UNKNOWN RANK OR SHAPE MUST BE MADE, AND, IF WE WERE TO USE SIMPLY = OR ≠ WE COULD END UP WITH AN ERROR MESSAGE OR A LOGICAL VECTOR, WHEN, IN FACT, WE WOULD LIKE A SOLITARY 1 (TRUE) OR 0 (FALSE).

YOU MIGHT THINK THAT THE ∧.= OR ∧.≠ INNER PRODUCT WOULD DO THE TRICK - WELL, IT WILL, PROVIDED THE INNER DIMENSIONS OF ITS ARGUMENTS ARE IDENTICAL OR SCALAR (MORE ON THAT LATER!).

FOR THE MOMENT, ASSUME THAT IT WILL NOT WORK. NOW, WRITE A FUNCTION WHICH WILL COMPARE ANYTHING TO ANYTHING AND YIELD AN EXPLICIT 1 OR 0, AND NEVER GENERATE AN ERROR MESSAGE (SUCH AS "RANK" OR "LENGTH").

AND HERE'S SOME HELP:

1. CALL YOUR FUNCTION "EQUALS"
2. CHECK, IN ORDER, FOR EQUALITY IN RANK, SHAPE, AND COMPONENTS

TEST YOUR FUNCTION WITH A VARIETY OF DATA, AND, WHEN
YOU ARE SATISFIED WITH IT, TYPE: FINISHED
AND I'LL CHECK IT OUT FOR YOU.

(IF YOU REALLY HAVE PROBLEMS FIGURING OUT WHERE TO BEGIN,
TYPE: GIVEUP
AND I'LL HELP YOU SOME MORE.)

```
      ⍴⍳0
0
      ∇E
       ∇
       C←A EQUALS B
[1]   D←(⍴⍴A)=⍴⍴B
[2]   →2
[3]   [1]
[1]   C←0
[2]   D←(⍴⍴A)=⍴⍴B
[3]   →4 0[2-D]
[4]   D←(⍴A)=⍴B
[5]   →6 0[2-D]
[6]   [5]
[5]   →6 0[2-D]
[6]   D←A=B
[7]   D←,D
[8]   C←∧/D
[9]   ∇
      A←2 3⍴⍳6
      B←2 3⍴⍳5
      A
 1  2  3
 4  5  6
      B
 1  2  3
 4  5  1
      A EQUALS B
0
      B←A
      A EQUALS B
1
      B←2 3 4 5 6⍴1
      A EQUALS B
0
      FINISHED
```

IF THIS FUNCTION INTERRUPTS, AND A SYSTEM ERROR MESSAGE
RESULTS, IT MEANS THAT YOUR FUNCTION IS NOT
DOING WHAT IT SHOULD.
SO CHECK FOR:

1. *DOES IT HAVE AN EXPLICIT RESULT?*
2. *DOES IT HANDLE NULL VECTORS?*
3. *IS THE FUNCTION NAME "EQUALS", AND IS IT DYADIC?*

THEN CLEAR THE STATE INDICATOR, RE-WRITE "EQUALS", AND WE'LL TRY AGAI
.

HERE GOES.....

VERY GOOD!
YOUR "EQUALS" FUNCTION WORKS WELL.
KEEP IT - IT'LL BE USEFUL.

LET'S LOOK AT MORE RIGOROUS DEFINITIONS OF INNER AND OUTER PRODUCTS.

OUTER PRODUCT:

```
      R[I;J] = A[I] (F) B[J]
```

AND: (ρR) = (ρA),ρB

WHICH IS PRETTY STRAIGHTFORWARD.

INNER PRODUCT:

REMEMBER I SAID I'D SAY SOME MORE ABOUT THE "INNER" DIMENSIONS LATER? WELL, NOW IS LATER!

CONSIDER 2 ARRAYS:

```
(ρA) = 2 3 4
(ρB) = 3 9 2
```

IN THIS CASE, ANY INNER PRODUCT INVOLVING "A" AND "B" WOULD GENERATE AN ERROR MESSAGE (CAN YOU GUESS WHICH ONE?) BECAUSE THE ARGUMENTS (A AND B) ARE NOT CONFORMABLE.

LET'S DEFINE THAT:

THE ARGUMENTS OF AN INNER PRODUCT ARE CONFORMABLE IF ANY OF THE FOLLOWING CONDITIONS ARE MET:

```
      1. (ρA)[ρρA] = (ρB)[1]
OR:   2. (1=(ρA)[ρρA])∨1=(ρB)[1]   IS TRUE
OR:   3. EITHER ARGUMENT IS SCALAR
```

FOR THE ARRAYS IN OUR EXAMPLE TO BE CONFORMABLE, EITHER
(ρA) = 2 3 3 OR (ρB) = 4 9 2
WOULD HAVE TO BE THE CASE.

REASON? EASY - THE INNER DIMENSIONS (THAT IS, THE RIGHT-MOST DIMENSION OF THE LEFT ARGUMENT, AND THE LEFT-MOST DIMENSION OF THE RIGHT ARGUMENT) ARE ELIDED FROM THE RESULT.

THUS, IN OUR EXAMPLE, THE SHAPE OF THE RESULT (GIVEN THAT (ρA) = 2 3 AND (ρB) = 3 9 2) WOULD BE: (ρR) = 2 3 9 2.

TO PUT THIS INTO A GENERAL FORM:

```
      (ρR) = (¯1↓ρA),1↓ρB
```

AND, FOR RANK 2 ARRAYS WHICH ARE CONFORMABLE, WE CAN SAY:

```
GIVEN: R← A (F1).(F2) B
THEN:  R[I;J] = (F1)/A[I;] (F2) B[;J]
```

AND THAT'S ALL I HAVE TO SAY ABOUT THAT!

THE NEXT MODULE IS "46 LESSON21", WHICH DISCUSSES SOME ADVANCED TECHNIQUES IN APL\360 USAGE.

```
      )LOAD 46 LESSON21
SAVED  18.40.46 10/06/75
      )WIDTH 70
WAS 120
      START
```


HERE'S A LIST OF TOPICS IN THIS WS.
TO STUDY ANY ONE, JUST ENTER ITS CODE:

CODE	SUBJECT
T1	ADVANCED BRANCHING TECHNIQUES
T2	COMMON (PUBLIC) LIBRARIES
T3	DATA TYPE CONVERSION
T4	SCANNING FOR SEPARATORS
T5	CAI PROGRAMMING TECHNIQUES
T6	CORE-SAVING TECHNIQUES
T7	AVAILABLE I-BEAM FUNCTIONS
T8	SYSTEM ERROR MESSAGES
T9	TRACING FUNCTION EXECUTION
T10	STOP CONTROL

WHEN YOU'VE FINISHED STUDYING THOSE
TOPICS WHICH INTEREST YOU, TYPE: DONE

T1

BRANCHING

IN LESSON 15 YOU LEARNED A FEW SIMPLE METHODS OF BRANCHING.
HERE, WE SHALL EXPAND UPON THEM - WITH EMPHASIS UPON CODING
MULTIPLE BRANCHES IN ONE STATEMENT.

EXAMINE THIS:

```
[?]    →(((0≠1|X)∨(X<1)∨X>99),(50=X),(40=X),35=X←1↑,[,0ρ[←'ENTER "X": '
/RGERR,G5,G4,G35
```

THIS STATEMENT DOES THE FOLLOWING:

1. DISPLAYS THE MESSAGE: ENTER "X":
2. DISPLAYS: [: AND WAITS FOR EVALUATED INPUT
3. RAVELS WHATEVER WAS ENTERED
4. TAKES THE FIRST COMPONENT OF THE RAVELLED INPUT AND ASSIGNS IT TO "X"
5. YIELDS A 1 OR 0 DEPENDING UPON WHETHER X IS 35 OR NOT
6. SAME AS (5) FOR X=40 AND X=50
7. YIELDS A 1 FOR ANY OF THESE CONDITIONS:
   A. X>99
   B. X<1
   C. X IS NOT AN INTEGER
8. USES THE RESULTING LOGICAL VECTOR (WHICH, INCIDENTALLY, IN THIS EXAMPLE COULD ONLY CONTAIN ONE "1") TO COMPRESS A VECTOR CONSISTING OF LINE LABELS (EACH LABEL HAS ITS LINE NUMBER AS ITS VALUE). IF THE LOGICAL VECTOR WERE 4ρ0 THE RESULT WOULD BE →⍳0 WHICH "FALLS THROUGH".

THUS, WE HAVE AN EXAMPLE OF A 4-WAY BRANCH, PLUS INPUT AND OUTPUT, ALL ON ONE LINE.

OF COURSE, THERE ARE DIFFERENT WAYS TO ACHIEVE THE SAME EFFECT IN ONE LINE.
THIS, FOR EXAMPLE:

```
[?]   →((∨/(0≠1|X),(X<1),X>99),(50=X),(40=X),35=X←1↑,⍞,0⍴⎕←'ENTER "X":
')/RGERR,G5,G4,G35
```

OR THIS:

```
[?]   →(~(∧/(0=1|X),(X≥1),X≤99),(50≠X),(40≠X),35≠X←1↑,⍞,0⍴⎕←'ENTER "X"
:')/RGERR,G5,G4,G35
```

AND WHICH ONE YOU SELECT WILL DEPEND UPON WHAT YOU WANT TO DO. HOWEVER, IT IS USEFUL TO REMEMBER THAT THE VERSION WHICH WILL EXECUTE THE FASTEST IS THAT ONE WITH THE LEAST NUMBER OF FUNCTIONS IN IT.

ANOTHER USEFUL TECHNIQUE CAN BE USED WHEN YOU WISH TO BRANCH TO, SAY, LINE [20] IF I=1, OTHERWISE YOU WANT TO BRANCH TO LINE [12].
SO:

```
[?]   →((I=1)/20),12
```

WILL DO THE JOB NICELY FOR US, BECAUSE IF I=1, THEN THE RIGHT ARGUMENT OF THE BRANCH ARROW BECOMES: 20 12, AND, SINCF IT IS AN UNINDEXED VECTOR, A BRANCH WILL OCCUR TO THE LINE NUMBER REPRESENTED BY THE FIRST COMPONENT OF THE VECTOR.

AND, IF I≠1, THEN THE VECTOR BECOMES: (⍳0),12 WHICH EVALUATES TO SIMPLY 12.

LET US SUPPOSE WE WERE FACED WITH THIS BRANCHING PROBLEM:

```
I = 0  1  2  3  4  5  6  7  (NONE OF THESE)
 → 5 13  5  5  8  9 28 34        44
```

WE COULD CODE:

```
[?]   →((I=¯1+⍳8)/5 13 5 5 8 9 28 34),44
```

SOMETIMES IT'S USEFUL TO SET UP A VECTOR OF LINES, EITHER BY ACTUAL NUMERICAL VALUE, OR BY LABEL. WE CAN THEN USE THE VALUE OF "I" TO ROTATE THE VECTOR FOR BRANCHING PURPOSES.

ASSUME THAT:

```
      LINEVECTOR←LAB1,LAB2,34,2,5,44,LAB6
```

AND THAT "I" COULD ONLY HAVE THE VALUES ¯1+⍳7
THEN, BY SETTING "I" AT SEVERAL POINTS IN OUR PROGRAM, WE NEED ONLY USE ONE BRANCH STATEMENT, THUS:

```
[?]   →I⌽LINEVECTOR
```

WHICH WILL MINIMIZE THE AMOUNT OF BRANCH CODING.

(BY THE WAY, THERE ARE SO MANY POSSIBLE CORRECT ANSWERS TO A PROBLEM ON BRANCHING, THAT IT BECOMES IMPRACTICAL TO TRY AND EVALUATE A RESPONSE WITHOUT ASKING THE STUDENT TO ENTER HIS ANSWER AS MANY TIMES AS THERE ARE BRANCH POINTS IN THE STATEMENT. SO, FOR THIS TOPIC, YOU WILL HAVE TO BE CONTENT WITH READING THE TEXT - SORRY.)

ANOTHER USEFUL TECHNIQUE INVOLVES INCREMENTING A COUNTER IN THE BRANCH STATEMENT ITSELF.

HERE ARE 5 DIFFERENT WAYS IT CAN BE ACCOMPLISHED (AND THERE ARE 20 MORE THAT I WON'T BOTHER YOU WITH!):

```
[?]   →END×⍳MAX=I←I+1

[?]   →(MAX=I←I+1)⍴END

[?]   →(MAX=I←I+1)/END

[?]   →END⌈⍳MAX=I←I+1
```

(WARNING: "⌈⍳" WORKS ONLY IF THE LEFT ARGUMENT IS NON-ZERO)

```
[?]   →(,END)[⍳MAX=I←I+1]
```

ETC., ETC., ETC.

WHICH BRINGS US TO THE END OF THIS DISCUSSION ON BRANCHING.

T2

COMMON LIBRARIES

SOMETIMES CALLED "PUBLIC" LIBRARIES, THESE ARE LIBRARIES WITH SPECIALLY ASSIGNED NUMBERS (USUALLY IN THE RANGE 1 TO 999 - BUT YOU'D BETTER CHECK THE SYSTEM YOU ARE USING, TO MAKE CERTAIN).

THE BEST-KNOWN COMMON LIBRARY IS LIB 1, WHICH CONTAINS (AT LEAST) THESE WORKSPACES:

APLCOURSE
NEWS
PLOTFORMAT

1 APLCOURSE IS SIMILAR TO LESSONS 3, 5, 8, AND 24 IN THIS COURSE, EXCEPT THAT YOU ARE GIVEN A CHOICE OF PROBLEM TYPES IN IT.

1 PLOTFORMAT CONTAINS A SERIES OF FUNCTIONS WHICH PROVE USEFUL FOR THOSE DESIROUS OF GENERATING GRAPHS AS OUTPUT.

1 NEWS CONTAINS (YOU WILL RECALL) SYSTEM STATUS DATA. YOU USE IT LIKE THIS:

AFTER YOU'VE LOADED IT, IF YOU WANT TO KNOW WHAT IS CONTAINED IN IT WHICH MAY BE OF INTFREST TO YOU, EXECUTE: INDEX
AND YOU WILL RECEIVE A LIST OF THE TOPICS AND THE DATES ON WHICH THEY WERE ENTERED. TO ACCESS A FULL PRINTOUT OF ANY TOPIC, EXECUTE: PRINT × (WHERE "×" REPRESENTS THE LINE NUMBER SHOWN IN THE INDEX PRINTOUT).

ANOTHER FEATURE OF 1 NEWS IS CALLED "APLNOW".
THIS IS USED BY EXECUTING: APLNOW × × × (WHERE "× × ×" IS A VECTOR RIGHT ARGUMENT REPRESENTING MONTH, DAY, AND YEAR.

NOW IT'S YOUR TURN -- DO THE FOLLOWING:

1. FIND OUT WHAT'S IN LIB 1

3. LOAD 1 NEWS AND SEE HOW IT WORKS
3. LOOK AT SOME OF THE OTHER WORKSPACES IN LIB1
(EACH WS IN PUBLIC DOMAIN SHOULD HAVE A "DESCRIBE" OR "HOW" VARIABLE OR FUNCTION WHICH YOU CAN EXECUTE TO TELL YOU ABOUT THE CONTENTS OF THAT WS.)

WHEN YOU'VE DONE THAT, RE-LOAD THIS MODULE, AND TYPE: COMMON1

SEE YOU LATER!

```
      )LIB 1
ADVANCED
APFNS
APLCOURS
APLFILE
CETEST
COIBM
CONVERSI
CONVERT
FEDIT
FORMAT
HOWEDITS
LHPRINT
NEWS
PLOTFORM
PRINT
REL2NEWS
SEDIT
SPECIAL
TYPEDRIL
WSFNS
XREF
      )LOAD 1 NEWS
SAVED  15.42.36 03/12/75
      )VARS
CHANGE11        CHANGE13        CHANGE14        CHANGE15        CHANG
CHANGE22        CHANGE23        CHANGE24        CHANGE25        CHANG
      )WIDTH 70
WAS 120
      )VARS
CHANGE11        CHANGE13        CHANGE14        CHANGE15
CHANGE17        CHANGE18        CHANGE20        CHANGE21
CHANGE22        CHANGE23        CHANGE24        CHANGE25
CHANGE27        CHANGE5 CHANGE6 CHANGE8 DESCRIBE        I       MDX
MSGS    NEWSMAKING      PTX     SKD     SKNT    CR      NS      SD
      DESCRIBE
```

THIS WORKSPACE PROVIDES INFORMATION ABOUT THE OPERATION AND USE OF APL. THE FUNCTIONS OF INTEREST TO THE USER ARE APLNOW, INDEX, PRINT, AND SCHEDULE.

APLNOW TAKES AS ITS SINGLE ARGUMENT A THREE-ELEMENT VECTOR REPRESENTING A DATE, AS MONTH, DAY, YEAR. APLNOW PRINTS NOTES ON THE STATUS OF THE APL SYSTEM; FOR INSTANCE, RECENTLY ADDED FEATURES, TEMPORARY RESTRICTIONS, OR ADVICE ON PROGRAMMING OR TERMINAL OPERATION. ONLY THOSE NOTES ENTERED INTO APLNOW ON OR AFTER THE DATE GIVEN AS AN ARGUMENT ARE PRINTED.

INDEX TAKES NO ARGUMENT. IT PRINTS INDICES, DATES, AND THE FIRST FEW WORDS OF EACH NOTE IN APLNOW.

PRINT TAKES AS ITS SINGLE ARGUMENT THE INDEX (AS INDICATED

BY THE INDEX FUNCTION) OF A NOTE FROM APLNOW, AND PRINTS THE NOTE.

SCHEDULE TAKES NO ARGUMENT. IT INDICATES THE REGULAR DAILY APL SCHEDULE, AND ALL ANTICIPATED DEVIATIONS FROM THE NORMAL SCHEDULE.

```
      )[
       v
       LOAD 1 APLNOW
WS NOT FOUND
      )FNS
APLNOW  CHANGELIST      CHANGES CHANGESUMMARY   CHANGE1 CHANGE10
CHANGE12        CHANGE16        CHANGE19        CHANGE2 CHANGE2
      APLNOW 1
             v
             7 1 1976
      APLNP
          v
          OW 1 1 1976
      APLNOW 1 1 1975
      APLNOW 1 1 76
      INDEX
 1    APL/CMS (LIKE APLSV), ALTHOUGH 07/04/74 1663
      )LOAD 46 LESSON
                    v
                    21
SAVED  18.40.46 10/06/75
      )WIDTH 70
WAS 120
      COMMON1
```

HELLO AGAIN!

I WOULD LIKE TO DISCUSS SOME OF THE OTHER COMMON LIBRARIES, BUT, SINCE I DON'T KNOW WHICH ONES THEY ARE IN ALL SYSTEMS, I SUGGEST YOU CONTACT YOUR SYSTEM ADMINISTRATOR AND FIND OUT FROM HIM.

WHICH BRINGS US TO THE END OF OUR TALK ON COMMON LIBRARIES.

T3

DATA TYPE CONVERSION

THERE ARE MANY TIMES (WHEN FORMATTING OUTPUT, ESPECIALLY!) WHEN YOU WOULD LIKE TO BE ABLE TO CONTROL THE NUMBER OF PRINTED INTEGER AND DECIMAL POSITIONS IN NUMERIC DATA.

UNFORTUNATELY, THERE IS NO PRIMITIVE FUNCTION, AS YET, WHICH WILL DO THIS FOR YOU. SO I WROTE ONE. (AS, NO DOUBT, DID MANY OTHERS.)

I HAVE A FUNCTION IN THIS WS CALLED "STL" (NOT A MNEMONIC!) WHICH IS DYADIC, WITH EXPLICIT RESULT. IT WORKS LIKE THIS:

THE RIGHT ARGUMENT IS ANY NUMERIC SCALAR OR VECTOR.
THE LEFT ARGUMENT IS A 2-COMPONENT NUMERIC VECTOR.
THE FIRST COMPONENT IS THE NUMBER OF POSITIONS YOU WANT EACH CONVERTED COMPONENT OF THE RIGHT ARGUMENT

TO OCCUPY IN THE EXPLICIT RESULT, AND THE SECOND COMPONENT IS THE NUMBER OF DECIMAL PLACES (ROUNDED) IN EACH CONVERTED RIGHT ARGUMENT FIELD.

IF THE FIRST COMPONENT OF THE LEFT ARGUMENT DOES NOT PROVIDE SUFFICIENT SPACE (AND IT MUST INCLUDE ROOM FOR A NEGATIVE SIGN, OR A DECIMAL POINT, IF NECESSARY) FOR A RESULT FIELD, THEN HIGH-ORDER TRUNCATION WILL OCCUR.

HERE'S AN EXAMPLE OF IT AT WORK:

```
      10 2 STL ¯3  0  4.56789
      -3.00      0.00      4.57
```

AND ANOTHER:

```
      5 0 STL ⍳5
    1    2    3    4    5
```

THE RANGE FOR DECIMAL PLACES IS 0 TO 10, AND A SCALAR LEFT ARGUMENT WILL DEFAULT TO 1 DECIMAL PLACE.

WHAT I'D LIKE YOU TO DO NOW IS TO:

1. *DISPLAY FUNCTION "STL"*
2. *EXECUTE IT SEVERAL TIMES, AND WHEN YOU'RE SATISFIED,*
3. *TYPE: CONVERT*1

AND WE'LL DISCUSS THE FLIP SIDE (LITERAL DATA TO NUMERIC).

```
      )STL[⎕]∇
    ∇ Z←A STL C;B;I;J;K;N;R
[1]   Z←⍳K←0×J←1+⍴C←,C,0⍴A←A,1
[2]   →((J=K),0=B←C[(⍴C)⌊K←K+1])/0 5,0⍴R←'0',((0≠A[2])/'.'),A[2]⍴'0'
[3]   B←B×0≠+/R←,(((I←1)⌈+/N≥0,10*⍳12)⍴10)⍳N←⌊0.5+|B×10*I←1⍴⌈/((0≠,(A
2]⍴10)⍳(⌊0.5+(1||B)×10*A[2]))/⍳A[2]),0
[4]   R←R,(0≠A[2])/((I=0)/'.'),(A[2]-(⍴R)-(⍴R)|(R←('0123456789.-')[1+
(B<0)/11),I⍴,((-I)⍴R),(I≠0)/10])⍳'.')⍴'0'
[5]   →2,⍴Z←Z,(-A[1])↑R
    ∇
      5 2 STL ⍳8
 1.00 2.00 3.00 4.00 5.00 6.00 7.00 8.00
      2 5 STL ⍳8
0000000000000000
      2 STL ⍳8
.0.0.0.0.0.0.0.0
      5 STL ⍳8
  1.0  2.0  3.0  4.0  5.0  6.0  7.0  8.0
      CONVERT1
```

NOW WE'LL TAKE A LOOK AT CONVERTING A LITERAL STRING INTO A SIGNED SCALAR.

I HAVE ANOTHER FUNCTION IN THIS WS CALLED "CVT", AND THIS FUNCTION IS MONADIC, WITH EXPLICIT RESULT. CHARACTERS NOT IN THE SET:

```
      ¯ - . 0 1 2 3 4 5 6 7 8 9
```

ARE COMPRESSED OUT OF ITS LITERAL ARGUMENT, AND THE BALANCE IS CONVERTED TO A SIGNED SCALAR.

A CORRECT RESULT WILL BE YIELDED ONLY IF:

1. THERE IS EXACTLY ONE '-' OR '¯'.
2. IT IS THE FIRST SET CHARACTER ENCOUNTERED.

EXAMPLES:

```
      CVT '¯567.3040'
¯567.304

      CVT '6'
6

      CVT '-.012'
¯0.012

      CVT 'ABCDE'
0
```

AND IT'S YOUR TURN ONCE AGAIN!

DISPLAY FUNCTION "CVT"; EXECUTE IT A FEW TIMES;
THEN TYPE: CONVERT2

```
      ∇CVT[[⎕]∇
    ∇ Z←CVT K;A;I
[1]   Z←10⊥(1 ¯1)[1+I]×((A⍳('.'≠K)/,K)-1)×10*('.'∈,K)×((,K)⍳'.')-⌽,K←
I←(1↑K)∈'¯-')↓K←(K∈(A←'0123456789'),'.¯-')/K
    ∇
      CVT 'TROEP'
0
      CVT'TROEP-.6'
¯0.6
      CVT 'EEN HOOP TEKST MET SLECHTS 2 CIJFERS IN 1 REGEL'
21
      CONVERT2
```

AND THAT ENDS OUR DISCUSSION OF DATA TYPE CONVERSION.

T4

SCANNING INPUT STREAMS FOR SEPARATORS

I HAVE A GENERAL PURPOSE FUNCTION IN THIS WS CALLED "POINT" WHICH IS DYADIC, WITH EXPLICIT RESULT.

IT DOES THE FOLLOWING:

YIELDS A VECTOR OF THE INDICES OF ALL OCCURRENCES OF ITS LEFT ARGUMENT IN ITS RIGHT ARGUMENT.

SIMPLE, NO? AND YET VERY USEFUL.

CONSIDER: INPUT CONSISTS OF LITERAL DATA WITH EACH FIELD SEPARATED BY ANY ONE OF: , . : ; / AND YOU WOULD

LIKE TO KNOW WHERE EACH FIELD ENDS AND THE NEXT ONE BEGINS.

OK, DO THIS:

```
      V←',.:;/' POINT M←⍞
```

AND YOU'RE IN BUSINESS!

DISPLAY FUNCTION "POINT" AND TRY IT OUT.
PLEASE ERASE ANY GLOBAL VARIABLES YOU MAY
USE, AND PLEASE USE ONLY ALPHABETIC NAMES.

WHEN YOU'VE DONE THAT, TYPE: STREAM1

```
      ∇POINT[⎕]∇
    ∇ R←A POINT B
[1]   R←(B∈A)/⍳⍴B
    ∇
      V←',.:;/' POINT M←⍞
HIER KOMT DAN EEN TEKST MET DE TEKENS , . : ; EN NATUURLIJK DE /
      V
39 41 43 45 64
      V←',:.;/\' POINT M←⍞
.........................
      V
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27
      STREAM
            ∨
            1
```

THERE REALLY ISN'T MUCH MORE TO SAY ABOUT SCANNING INPUT STREAMS.
LESS SOPHISTICATED TECHNIQUES (SUCH AS DYADIC ⍳) MAY ALSO BE USED
AND SHOULD BE IF THEY PROVIDE THE DESIRED RESULTS.

AND THAT'S THAT!

T5

CAI PROGRAMMING TECHNIQUES

IN THIS SECTION, WE SHALL NOT BE CONCERNED WITH WHAT IS GOOD
CAI PRACTICE; RATHER, WE SHALL LOOK AT SOME FUNCTIONS WHICH
I HAD TO WRITE TO SIMPLIFY WRITING THIS COURSE, AND WHICH
YOU MAY FIND USEFUL.

THE FIRST FUNCTION WE SHALL DISCUSS IS CALLED: "NO"
IT IS NILADIC, WITH EXPLICIT RESULT.
IT WAITS FOR LITERAL INPUT, AND, IF THE FIRST
CHARACTER OF THE INPUT IS "N", YIELDS A 1.
IF THE FIRST INPUT CHARACTER IS A "Y", IT YIELDS A 0.
IF NEITHER, IT PRINTS THIS MESSAGE: YES OR NO?
AND WAITS FOR INPUT AGAIN.

USING THIS FUNCTION WILL AVOID MUCH REPETETIVE CODING IN
CAI MATERIAL IN APL\360, AND HERE ARE SOME EXAMPLES OF ITS USE:

```
[?]   →NO/NEXT,0⍴⍞←'SHALL WE LOOK AT THE NEXT TOPIC?'

[?]   →(NO/0),0⍴⍞←'MORE?'

[?]   →(~NO)⍴0⍴⍞←'FINISHED?'
```

```
AND SO ON.

NOW, YOU DISPLAY FUNCTION "NO",
EXECUTE IT A FEW TIMES, THEN TYPE: CAI1

      ∇NO[□]∇
    ∇ R←NO;J;K
[1]   →1×J,(R←K/0 1),ρ⍞←(J←~1∈K←'YN'=1↑,⍞)/'YES OR NO?'
    ∇
      NO
DEZE TEKST BEGINT NIET MET N OF ↑
                                ∨
                                Y, DUS MOET DE MELDING YES OR NO WORD
YES OR NO?
NEE

1
      NO
↑AWEL
YES OR NO?
YAWEL

0
      CAI1

ANOTHER PROBLEM WHICH FREQUENTLY CROPS UP WHEN DEALING
WITH INPUT FROM STUDENTS WITH A WIDE VARIETY OF BACKGROUNDS
IS THAT OF CORRECT SPELLING OF A PLAIN TEXT TYPE OF INPUT.

SO, TO MAKE LIFE EASIER, THE FUNCTION "CA" (CORRECT ANSWER)
WILL DO THE FOLLOWING:

YIELD A LITERAL STRING OF EITHER THE ENTIRE WORD, WITH
LEADING AND TRAILING BLANKS REMOVED, PROVIDED ONLY ONE
WORD WAS IN THE INPUT,

                    OR

YIELD A LITERAL CONSISTING OF THE FIRST CHARACTER OF
EACH WORD IN THE INPUT.

FOR EXAMPLE:

     CA '   INTERNATIONAL     BUSINESS   MACHINES   '
IBM

     CA '    HELP  '
HELP

AND IS GENERALLY USED THUS:

[?]   →(('HELP'∧.=4↑,X),'ANSWER'∧.=6↑,X←CA⍞)/H1,C1,0ρ⍞←'WHAT AM I LOO
ING FOR?'

DISPLAY THE FUNCTION, TRY IT, THEN TYPE:  CAI2

      ∇CA[⎕]∇
    ∇ R←CA X;I;J;V
[1]   R←J[((0=ρV)/ιρJ),(0≠ρV)/1,1+V←(J∈' ')/ιρJ←((J[1]≠' '),((¯2+ρJ)ρ
),J[ρJ]≠' ')/J←((1ϕI)≠I←X∈' ')/X←' ',X]
    ∇
      CA 'BADENSE ANILINE EN SODA FABRIEK'
BAFSF
      CA 'BADENSE ANILINE-EN SODA FABRIEK'
```

```
BASF
      CA ' EENWOORD'
EENWOORD
      CAI2
```

IF YOU WISHED TO CHECK A COMPLETE RESPONSE, CONSISTING OF SEVERAL WORDS (SUCH AS IN A SPELLING QUIZ, WHERE THE STUDENT HAS TO ENTER A STRING OF WORDS AS LITERAL INPUT) THERE ARE 2 MORE FUNCTIONS YOU COULD USE.

THE FIRST, CALLED "SQZ", IS A SUBSET OF FUNCTION "CA", AND DOES THIS:

1. REMOVES ALL LEADING AND TRAILING BLANKS
2. REDUCES THE NUMBER OF BLANKS SEPARATING NON-BLANK STRINGS TO 1
3. YIELDS WHAT'S LEFT AS AN EXPLICIT RESULT

SO:

```
      SQZ '   INTERNATIONAL  BUSINESS    MACHINES  '
INTERNATIONAL BUSINESS MACHINES
```

THE SECOND FUNCTION, "DBL", MERELY REMOVES ALL BLANKS FROM ITS ARGUMENT, AND YIELDS THE BALANCE AS A LITERAL STRING.

DISPLAY THEM, TRY THEM, THEN TYPE: CAI3

```
      DBL 'L A A T   U   M A A R   Z I T T E N '
LAATUMAARZITTEN
      ∇SQZ[[]]'
DEFN ERROR
      ∇SQZ[[]]'
           ^
      G
      ∇
      ∇SQZ[[]]'
DEFN ERROR
      ∇SQZ[[]]'
           ^
      ∇SQZ[[]]∇
    ∇ R←SQZ X;I
[1]   R←((I[1]≠' '),((¯2+ρI)ρ1),I[ρI]≠' ')/I←((1⌽I)≠I←X∈' ')/X←' ',X
    ∇
      G
      ∇
      ∇DBL[[]]∇
    ∇ R←DBL X
[1]   R←(X≠' ')/X
    ∇
      CAI3
```

A USEFUL FUNCTION IS CALLED "SCAN".
IT IS MONADIC, WITH EXPLICIT RESULT.
IT DOES THIS:

1. REMOVES ALL LEADING AND TRAILING BLANKS FROM ITS LITERAL ARGUMENT
2. REDUCES THE NUMBER OF BLANKS SEPARATING NON-BLANK STRINGS TO 1
3. SPECIFIES A GLOBAL VARIABLE, I, TO BE THIS "SQUEEZED" ARGUMENT
4. YIELDS AN EXPLICIT RESULT OF THE INDICES (IN "I") OF THE FIRST CHARACTER OF EACH NON-BLANK STRING

DISPLAY THE FUNCTION, TRY IT OUT, PLEASE ERASE I WHEN YOU'RE THROUGH,

AND THEN TYPE: CAI4

```
      ∇SCAN[□]∇
    ∇ R←SCAN X
[1]   R←1,1+(I∈' ')/⍳⍴I←((I[1]≠' '),((¯2+⍴I)⍴1),I[⍴I]≠' ')/I←((1ΦI)≠I
X∈' ')/X←' ',X
    ∇
      SCAN 'DE VOLGENDE TEKST WORDT NU       WAT VERKNOEID    '
1 4 13 19 25 28 32
      I
DE VOLGENDE TEKST WORDT NU WAT VERKNOEID
      )ERASE I
      CAI4
```

THAT ENDS OUR DISCUSSION OF USEFUL CAI FUNCTIONS.

CORE-SAVING TECHNIQUES

SOME OF THE THINGS YOU CAN DO TO MINIMIZE THE AMOUNT OF MEMORY USED IN EACH WS FOR DATA (OR FUNCTIONS) ARE:

1. FUNCTION AND VARIABLE NAMES

IF YOU LIMIT NAMES (AND THIS INCLUDES LINE LABELS!) TO 3 CHARACTERS APL WILL STORE THE NAME AS IS (I.E., WITHOUT A "POINTER") IN ITS SYMBOL TABLE. LONGER NAMES HAVE AN ADDITIONAL 5 BYTES (MINIMUM) OF OVERHEAD.

2. FUNCTION SIZE

USE ANY METHODS TO MINIMIZE THE NUMBER OF LINES IN A FUNCTION - THERE IS A 4-BYTE OVERHEAD PER LINE! ESPECIALLY BAD IS USING UP 1 LINE FOR AN EMPTY VECTOR FOR LINE FEEDING, WHEN THE CHARACTER "CR" (CARRIAGE RETURN) COULD BE CATENATED BETWEEN LITERAL OUTPUT STRINGS.

3. TEXT

WHENEVER YOU HAVE A TEXT TO BE OUTPUTTED WHICH CONSISTS OF MORE THAN 200 CHARACTERS, MAKE IT A VARIABLE, AND DISPLAY THAT VARIABLE DURING FUNCTION EXECUTION. YOU'LL SAVE AT LEAST 4 BYTES PER LINE, AND, IF THE TEXT IS CONSIDERABLE (LIKE IN THIS MODULE) THE SAVINGS CAN BE WELL WORTHWHILE.

4. DATA TYPE

HOW YOU ELECT TO SPECIFY DATA IS REALLY UP TO YOU.
BUT, BEAR THIS IN MIND:

LOGICAL DATA (I.E. 1'S AND 0'S) OCCUPY ONE BIT PER DIGIT.
INTEGERS OCCUPY 4 BYTES PER VALUE.
FRACTIONS OCCUPY 8 BYTES PER VALUE.

NOTE: THE OVERHEAD FOR ARRAY SPECIFICATION IS APPROXIMATELY THE SUM OF THE OVERHEAD FOR THE ARRAY NAME, PLUS UP TO 5 BYTES PER ROW (USUALLY FEWER).

AND THAT'S ALL I HAVE TO SAY ABOUT CORE-SAVING TECHNIQUES.

T7

I-BEAM FUNCTIONS

APL HAS CERTAIN SYSTEM FUNCTIONS WHICH ARE NILADIC, WITH EXPLICIT RESULTS, AND THEY TAKE THIS GENERAL FORM:

⌶*N* (WHERE "*N*" IS: 19≤*N*≤29)

THE SYMBOL ⌶ IS FORMED BY UPPER CASE "N", BACKSPACE, THEN UPPER CASE "B" - THAT IS: ⊤ OVERSTRUCK BY ⊥.

HERE'S A LIST OF THE 11 FUNCTIONS AVAILABLE TO YOU, WITH A BRIEF DESCRIPTION OF WHAT EACH DOES:

FUNCTION	PURPOSE
⌶19	ACCUMULATED KEYING TIME IN 60THS OF A SECOND DURING THIS SESSION, I.E., THE TOTAL TIME FOR WHICH THE KEYBOARD WAS UNLOCKED AWAITING ENTRIES.
⌶20	TIME OF DAY, IN 60THS OF A SECOND - THIS, BY THE WAY, IS ONLY AS ACCURATE AS THE CLOCK USED BY THE OPERATOR TO SET THE TIMER AT THE START OF THE WORKING DAY.
⌶21	TOTAL AMOUNT OF CPU TIME USED SINCE SIGN-ON, AGAIN, IN 60THS OF A SECOND.
⌶22	THE AMOUNT OF UNUSED MEMORY (IN BYTES) IN THE ACTIVE WS.
⌶23	THE NUMBER OF SUBSCRIBERS CURRENTLY ON THE SYSTEM.
⌶24	YOUROSIGN-ON TIME, IN 60THS OF A SECOND.
⌶25	CURRENT DATE - MMDDYY IN BASE 10.
⌶26	WE COVERED THIS IN LESSON 15, "BRANCHING". YOU WILL RECALL THAT IT IS THE CURRENT VALUE OF THE LINE COUNTER - THAT IS, IT CONTAINS THE LINE NUMBER OF THE LINE BEING EXECUTED.
⌶27	THIS IS THE VECTOR OF LINE NUMBERS IN THE STATE INDICATOR. THUS, ⍴⌶27 WILL TELL YOU HOW MANY ITEMS ARE IN THE STATE INDICATOR.
⌶28	YIELDS A DIGIT FROM THE SET ⍳4, REPRESENTING THE TERMINAL DEVICE YOU ARE USING, AND THEY MEAN: 1 - 2741 ATS 0 2 - 2741 TSS 3 - 1050 4 - CONSOLE TYPEWRITER
⌶29	YOUR SIGN-ON NUMBER

TO GIVE YOU AN IDEA OF HOW TO USE THESE SYSTEM FUNCTIONS, PARTICULARLY THOSE WHICH YIELD RESULTS IN 60THS OF A SECOND,

I'VE DEFINED 2 FUNCTIONS IN THIS WS:

DATE - YIELDS A LITERAL VECTOR REPRESENTING: MM/DD/YY
TIME - YIELDS A LITERAL VECTOR REPRESENTING: HH:MM:SS

WHAT I'D LIKE YOU TO DO IS:

1. *DISPLAY THEM*
2. *EXECUTE THEM*

THEN TYPE: IBEAM1
TO CONTINUE.

```
      ∇DATE[⎕]∇
    ∇ R←DATE
[1]   R←'/0123456789'[1+1 1 0 1 1 0 1 1\1+(6⍴10)⊤⌶25]
    ∇
      ∇TIME[⎕]∇
    ∇ R←TIME
[1]   R←':0123456789'[1+1 1 0 1 1 0 1 1\1+0 10 6 10 6 10⊤⌈(⌶20)÷60]
    ∇
      DATE
10/04/76
      TIME
11:51:52
      IBEAM1
```

BECAUSE THESE 2 FUNCTIONS YIELD LITERAL RESULTS, THEY CAN BE CATENATED INTO AN OUTPUT STREAM WHERE SUCH DATA MAY BE DESIRED.

YOU CAN FOOL AROUND WITH THE OTHERS AT YOUR LEISURE, BUT REMEMBER - "⌶" IS A PRIMITIVE FUNCTION; ITS RIGHT ARGUMENT CAN BE COMPUTED, BUT IF IT'S NOT IN THE DOMAIN OF THE FUNCTION, WELL.....!

AND THAT CONCLUDES THIS PRESENTATION ON I-BEAM FUNCTIONS.

SYSTEM ERROR MESSAGES

THIS TOPIC WILL CONSIST OF A LIST OF SYSTEM ERROR MESSAGES, THEIR CAUSES, AND THE CORRECTIVE ACTION REQUIRED TO RESUME.

MESSAGE	CAUSE	RECOVERY
CHARACTER ERROR	IMPROPER OVERSTRIKE	RE-ENTER
DEPTH ERROR	EXCESSIVE USE OF DEFINED FUNCTIONS WITHIN DEFINED FUNCTIONS	CLEAR THE STATE INDICATOR REPEATEDLY, AND, IF THIS FAILS, SAVE, CLEAR, COPY.
DEFN ERROR	1. SYNTAX ERROR IN FUNCTION HEADER 2. FUNCTION IS LOCKED 3. THAT NAME ALREADY USED IN WS 4. FUNCTION IS PENDENT	CORRECT OR RE-NAME THE FUNCTION; IT IS NOT POSSIBLE TO EDIT LOCKED FUNCTIONS. YOU CAN'T EDIT PENDENT FUNCTIONS, EITHER.
DOMAIN ERROR	ARGUMENTS NOT IN THE DOMAIN OF THE FUNCTION - OFTEN THE RESULT OF TRYING TO CATENATE LITERAL TO CONSTANT DATA.	RE-WRITE IT.
INDEX ERROR	REQUEST FOR A NON-EXISTENT ARRAY ELEMENT	CHECK THE ORIGIN, OR USE ⌈ OR ⌊ TO LIMIT INDEX RANGE
LENGTH ERROR	ARGUMENTS NOT CONFORMABLE	CHECK RANK AND SHAPE IN YOUR FUNCTION
RANK ERROR	FUNCTION NOT DEFINED FOR THIS STRUCTURE	RE-FORMULATE THE COMMAND.
SI DAMAGE	EDITING LABEL LINES, OR CHANGING THE NUMBER OF LINES IN A SUSPENDED FUNCTION.	CLEAR STATE INDICATOR; RE-EXECUTE AFTER EDITING.
SYMBOL TABLE FULL	TOO MANY NAMES USED.	SAVE, CLEAR, EXECUTE:)SYMBOLS N (256≤N≤4000), COPY, WSID.
SYNTAX ERROR	INCORRECT USE OF LANGUAGE STRUCTURE, MOST COMMONLY CAUSED BY UNBALANCED PARENTHESES, OR OMITTED CATENATIONS.	CORRECT, THEN RESUME EXECUTION.
SYSTEM ERROR	INDETERMINABLE SYSTEM FAULT	NONE - A CLEAR WS IS AUTOMATICALLY LOADED. THIS IS A GOOD REASON FOR FREQUENT SAVING BETWEEN FUNCTION EDITTING.
VALUE ERROR	1. NO PREVIOUSLY SPECIFIED VALUE FOR THE VARIABLE. 2. EXPLICIT FUNCTION RESULT NOT DEFINED DURING EXECUTION.	SPECIFY A VALUE, THEN RESUME EXECUTION. IN THE CASE OF EXPLICIT FUNCTION RESULT, RE-WRITE FUNCTION.
WS FULL	INSUFFICIENT MEMORY AVAILABLE IN WS, USUALLY CAUSED BY ∘ ↑ ↓ , ⍉ ⌽ / / OPERATIONS ON DATA.	DELETE UNNEEDED OBJECTS AND RESUME EXECUTION, OR RE-WRITE TO USE LESS CORE.

AND THAT'S ALL ON SYSTEM ERROR MESSAGES.

```
)WIDTH 70
WAS 120
      T9
```

TRACING FUNCTION EXECUTION

0

THE TRACE FUNCTION HAS BEEN PROVIDED IN APL AS A DE-BUGGING AID, AND IT IS SET UP LIKE THIS:

```
      TΔFNAME←2 17 4 66
```

WHICH WILL RESULT IN THE VALUES OF COMMANDS ON LINES 2 17 4 AND 66 (USUALLY THESE ARE BRANCH COMMANDS) BEING DISPLAYED DURING EXECUTION.

THE STATEMENT:

```
      TΔFNAME←0
```

WILL DISCONTINUE THE TRACE.

HERE'S AN EXAMPLE OF THE TRACE AT WORK ON A SIMPLE FUNCTION:

```
      ∇B←F1 X
[1]   B←1
[2]   B←(B,0)+0,B
[3]   →2×X>B[2]∇

      TΔF1← 2 3

      F1 3
F1[2] 1  1
F1[3] 2
F1[2] 1  2  1
F1[3] 2
F1[2] 1  3  3  1
F1[3] 0
1  3  3  1

      TΔF1←0
```

IT IS ALSO POSSIBLE TO SET THE TRACE FUNCTION ON A LINE WITHIN THE DEFINED FUNCTION ITSELF.

FOR EXAMPLE, THE STATEMENT:

```
[?]   TΔFNME←5 12 23×3<I←I+1
```

WILL CAUSE LINES 5 12 AND 23 TO BE TRACED ONLY AFTER "I" HAS A VALUE OF 4 (I.E., THE 4TH ITERATION, AND ALL SUCCEEDING ITERATIONS). OR OTHER VARIATIONS YOU CAN DREAM UP.

THE TRACE FUNCTION, CORRECTLY CALLED THE TRACE VECTOR, IS NOT A VARIABLE IN THE USUAL SENSE OF THE WORD - IT CANNOT BE EXAMINED OR COPIED.

DELETION OF A FUNCTION WITH THE TRACE VECTOR SET ALSO DELETES ITS TRACE CONTROL. EDITING A LINE WILL ALSO REMOVE THE TRACE CONTROL FOR THAT LINE.

5-

AND THAT'S L

T10

STOP CONTROL

THIS IS SIMILAR IN CONCEPT TO THE TRACE CONTROL, EXCEPT THAT INSTEAD OF "T", YOU USE "S".

ALSO, IT WILL SUSPEND EXECUTION IMMEDIATELY PRIOR TO EXECUTION OF A LINE WITH A STOP CONTROL.

AS FOR TRACE, STOP CONTROLS MAY BE SET WITHIN FUNCTIONS AND BEHAVE, AND ARE CLEARED IN THE SAME MANNER. THE ADVANTAGE OF THE STOP CONTROL OVER THE TRACE CONTROL IS THAT YOU CAN DISPLAY LOCAL VARIABLES DURING SUSPENSION OF THE FUNCTION.

EXECUTION IS RESUMED BY BRANCHING TO THE LINE INDICATED.

AND THAT'S IT FOR STOP CONTROL.

```
      )LOAD 46 LESSO22M
SAVED  13.37.05 02/17/75
      )WIDTH 70
WAS 120
      START
```

LESSON 22M

WE HAVE ALREADY LEARNED ABOUT THE MONADIC USE OF THE CIRCLE (RECALL THAT: ○N = (22÷7) × N) SO WE SHALL NOW EXAMINE THE DYADIC USE OF THE SAME CIRCULAR FUNCTION.

THE GENERAL FORM OF DYADIC "○" IS:

```
      A○B
```

WHERE:

A - IS AN INTEGER IN THE DOMAIN: ¯7≤A≤7
AND: B - IS THE VALUE ON WHICH THE FUNCTION IS TO OPERATE

THUS, THERE ARE 15 POSSIBLE FUNCTIONS AVAILABLE, AND HERE THEY ARE:

LEFT ARGUMENT	RESULT	DOMAIN RESTRICTIONS ON RIGHT ARGUME
¯7	ARCTANH	1>|B
¯6	ARCCOSH	B≥1
¯5	0 ARCSINH	
¯4	(¯1+B*2)*0.5	1≤|B
¯3	ARCTAN	
¯2	ARCCOS	1≥|B
¯1	ARCSIN	1≥|B
0	(1-B*2)*0.5	B≤1
1	SINE	
2	COSINE	
3	TANGENT	
4	(1+B*2)*0.5	
5	SINH	
6	COSH	
7	TANH	

LEFT ARGUMENTS OF ¯4, 0, AND 4 ARE NOT CIRCULAR FUNCTIONS IN A PURIST SENSE, BUT THEY ARE THERE BECAUSE THEY WERE NEEDED IN ORDER TO PROVIDE SOME OF THE OTHER CIRCULAR FUNCTIONS, AND THE DEVELOPERS OF APL THOUGHT THEY COULD BE USEFUL.

THERE IS ONE MORE RESTRICTION - THE RIGHT ARGUMENT MUST BE EXPRESSED IN RADIANS.

SO HERE'S A PROBLEM FOR YOU:

WRITE A FUNCTION WHICH WILL CONVERT DEGREES TO RADIANS.
CALL YOUR FUNCTION: DTR
AND MAKE IT MONADIC, WITH EXPLICIT RESULT.

WHEN YOU'VE DONE THAT, TYPE: DONE
AND I'LL CHECK IT FOR YOU.

1-

```
      ∇DTR
       ∨
       R←DTR D
[1]    R←○(D÷180)∇
      DTR 3.1415
          ∨
          180
3.141592654
      DONE
I'M CHECKING IT NOW.....

YOUR "DTR" FUNCTION YIELDS AN INCORRECT RESULT.
TRY AGAIN, OR TYPE: HELP 1

      ∇DTR[□]∇
    ∇ R←DTR D
[1]   R←○(D÷180)
    ∇
      ∇DTR[1□]
[1]   R←○(D÷180)
[1]   R←(4×¯301)×(1
                  ∨
                  D÷180)
[2]   ∇
      DTR 180
3.141592654
      DONE
I'M CHECKING IT NOW.....

GOOD!
YOUR "DTR" FUNCTION CHECKS OUT OK.

BUT - THAT'S ONLY HALF THE PROBLEM.....

NOW WRITE A FUNCTION, RTD, WHICH CONVERTS RADIANS TO DEGREES.
ONCE AGAIN, IT MUST BE MONADIC, WITH EXPLICIT RESULT.

WHEN YOU'RE FINISHED, TYPE: CHECK
AND I'LL CHECK THAT ONE FOR YOU.

    K*'IK ZIE GEEN VERSCHIL TUSSEN DE UITKOMSTEN VAN DE VERSCHILLEND

∨

SYNTAX ERROR
      ←
      ^
      ∇D←RTD R
[1]   D←180
        ∨
        (180÷4×¯301)×RV
      RTD 3.141592653590
180
      )DIGITS 16
WAS 10
      RTD DTR 180
180
      )DIGITS 10
WAS 16
      CHECK
CHECKING IT NOW.......

GREAT!

YOUR "RTD" FUNCTION YIELDS A CORRECT RESULT.
```

```
MAKE A NOTE OF THESE FUNCTIONS - YOU'LL FIND
THEM QUITE USEFUL IN THE FUTURE.
FOR NOW, WE'VE FINISHED, SO PROCEED TO MODULE
"46 LESSON23T" FOR THE FINAL TEST.

      ○1
3.141592654
      )DIGITS 16
WAS 10
      ○1
3.141592653589793
      'DIT IS DUS PI, EN NIET 22÷7'
DIT IS DUS PI, EN NIET 22÷7
```

```
      )LOAD 46 LESSO23T
SAVED  13.37.21 02/17/75
      )WIDTH 70
WAS 130
      START
ENTER YOUR NAME:

W . JANSSEN

THANK YOU.
```

LESSON 23T - FINAL TEST

A. EXAMINE THE FOLLOWING FUNCTION,
THEN ANSWER THE QUESTIONS:

```
      ∇F[[⎕]]∇
    ∇ F;J;Y
[1]   →(1 9 =J←⍴(Y∈'/')/⍳⍴Y←,⍞,'/')/END,2+⌶26
[2]   →1,⍴⍞←'WRONG - TRY IT AGAIN:'
[3]   →0=⍴⍞←'CORRECT!'
[4]  END:''
[5]   →0×⍳'Y'=1↑,⍞,0⍴⍞←'THAT'S NO ANSWER!  TRY ANOTHER?'
[6] 0 'WELL, ANOTHER TIME, PERHAPS.....'
    ∇
```

THE SOLE PURPOSE OF FUNCTION "F" IS TO TEST YOUR
UNDERSTANDING OF APL\360 CODING.

THERE ARE 5 QUESTIONS ON FUNCTION "F" IN THIS
PART (SECTION "A") OF THE TEST, AND YOU WILL BE
PERMITTED 3 ATTEMPTS AT EACH QUESTION, WITH FIRST-
TIME-CORRECT ANSWERS CARRYING MORE WEIGHT THAN
SECOND-TIME-RIGHT ANSWERS, ETC.

HERE WE GO.....

1. WHICH LINE WILL BE EXECUTED AFTER LINE [1]
   IF "J" EVALUATES TO 9?

```
[⎕]:
      3
```

RIGHT!

2. LINE [5] OF "F" HAS AN ERROR IN ITS BRANCH POINT.
   WHAT SHOULD IT BE?

```
[⎕]:
      1
```

VERY GOOD!

3. WHICH LINE GETS EXECUTED AFTER LINE [2]?

```
[⎕]:
      1
```

GOOD!

4. *ASSUME THAT THE INPUT ENTERED ON LINE [1] WAS:*

HELP/ME/OUT/PLEASE

WHAT WILL BE THE VALUE OF "J" IN LINE [1]?

```
[⎕:
      4
```

WELL DONE!

5. *INSTEAD OF ENTERING DATA ON LINE [1], YOU HIT "RETURN".*
WHAT LINE WILL BE EXECUTED NEXT?

```
[⎕:
      4
```

CORRECT!

THAT COMPLETES SECTION "A".

B. IN THIS SECTION, YOU HAVE TO WRITE A CONVERSATIONAL FUNCTION CALLED: ∇ GUESS WHICH DOES THE FOLLOWING:

1. *REQUESTS USER INPUT OF AN INTEGER IN THE RANGE: 1 TO 100*
2. *CALCULATES ITS SQUARE, DIVIDES BY 2, AND ADDS 50*
3. *SPECIFIES AND DISPLAYS A GLOBAL VARIABLE, A1, TO BE THE RESULT OF STEP (2) ABOVE.*
4. *SPECIFIES AND DISPLAYS ANOTHER GLOBAL VARIABLE, A2, TO BE "PI" (MONADIC CIRCULAR FUNCTION) TIMES THE SQUARE ROOT OF A1*

YOU SHOULDOBE ABLE TO CODE FUNCTION "GUESS" IN 1 LINE!
WHEN YOU'VE DEFINED IT, AND ARE SATISFIED THAT IT WORKS AS IT SHOULD, SIMPLY TYPE:

CONTINUE

AND I'LL CHECK IT OUT FOR YOU.

OFF YOU GO.....

```
      ∇GUESS
[1]   A2←○(A1←50+÷2×(P←⎕)*2)*.5⋄CR⋄A1⋄CR⋄A2∇
      ∇GUESS[⎕]∇
    ∇ GUESS
[1]   A2←○(A1←50+÷2×(P←⎕)*2)*0.5⋄CR⋄A1⋄CR⋄A2
    ∇
      )ERASE GUESS
      ∇GUESS
[1]   A2←○(A1←50+÷2×(P←⎕,0⍴(⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)


      ∇
      ∇
      GUESS
GEEF EEN INTEGER TUSSEN 1 EN 100
⎕:
      2
22.24216538
50.125
22.24216538
```

```
      ∇GUESS[⎕]∇
    ∇ GUESS
[1]   A2←⍕((A1←50+÷2×(P←⎕,0⍴(⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)*
0.5);CR;A1;CR;A2
    ∇
      )ERASE GUESS
      ∇GUESS
[1]   A2←(
          ⍕
        ⍕((A1←50+÷2×(P←⎕,0⍴(L←
                              ⍕
                            ⎕←'GEEF EEN INTEGER TUSSEN 1 W
                                                       ⍕
                                                      EN 100'))*2
[2]   ∇
      ∇GUESS[⎕]∇
    ∇ GUESS
[1]   A2←⍕((A1←50+÷2×(P←⎕,0⍴(⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2
*0.5);CR;A1;CR;A2
    ∇
      GUESS
GEEF EEN INTEGER TUSSEN 1 EN 100
⎕:
      2
22.24216538
50.125
22.24216538
      )ERASE GUESS
      ∇GUESS
[1]   A2←⍕((A1←50+(P←⎕,0⍴(⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)÷2
                  ⍕
                  (P←⎕,0⍴(L
                          ⍕
                        ⎕←'GEEF EEN INTEGER R
                                           ⍕
                                          TUSSEN 1 EN 100'))*2)÷
SYNTAX ERROR
[1]   A2←⍕((A1←50+([P←L,0R[⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)÷
                                                  )                ^
)*.5);CR;A1;CR;A2∇
[2]   ∇
      ∇GUESS[1⎕20]
[1]   A2←⍕((A1←50+([P←L,0 R[⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)÷
2)*.5);CR;A1;CR;A2∇
[1]   ∇
      )WIDTH 120
WAS 70
      ∇GUESS[1⎕20]
[1]   A2←⍕((A1←50+([P←L,0 R[⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)÷
      0             1/
[1]   A2←⍕((A1←50+(([←L,0 R[⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)
SYNTAX ERROR
[1]   A2←⍕((A1←50+(([←L,0 R[⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)
                                                                  ^
[2]   ∇
```

```
      VGUESS [1⎕20]
[1]   A2←○((A1←50+(([←L,0 R[⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)÷
                  /1
[1]   A2←○((A1←50+((P←L,0 R[⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)÷
SYNTAX ERROR
[1]   A2←○((A1←50+((P←L,0 R[⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)÷
                                                              ^
[2]   V
      )ERASE GUESS
      0VGUESS
[1]   A2←○((A1←50+(P←⎕,0ρ(⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)÷2
SYNTAX ERROR
[1]   A2←○((A1←50+(P←⎕,0ρ(⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)÷2

[2]   [1]
[1]   A2←○((A1←50+((P←⎕,0ρ(⎕←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)÷
      GUESS
GEEF EEN INTEGER TUSSEN 1 EN 100
⎕:
      2
22.6543468
52
22.6543468
      VGUESS[1⎕100]
[1]   A2←○((A1←50+((P←⎕,0ρ([←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)÷

[1]   A2←○((A1←50+((P←⎕,0ρ([←'GEEF EEN INTEGER TUSSEN 1 EN 100'))*2)÷
[2]   V
      GUESS
GEEF EEN INTEGER TUSSEN 1 EN 100
⎕:
      2
      CONTINUE


EXECUTE YOUR "GUESS" FUNCTION ONCE MORE,
THEN TYPE: DONE
TO PROCEED.

      GUESS
GEEF EEN INTEGER TUSSEN 1 EN 100
⎕:
      25
      DONE
WHAT NUMBER DID YOU ENTER?
⎕:
      25

VERY GOOD!  YOU HAVE THEM BOTH CORRECT.


WE'RE NOW READY FOR PART "C" (THE LAST SECTION).

C. THIS SECTION CONSISTS OF 4 SIMPLE QUESTIONS - SO
SIMPLE, IN FACT, THAT YOU'RE ONLY ALLOWED 1 CRACK AT EACH.

1. WHAT CHARACTER WOULD YOU USE TO UNDERSTRIKE
   ONE TO BE DELETED DURING FUNCTION EDITING?

/

CORRECT!
```

• *YOU'VE DISCOVERED AN ERROR DURING EXECUTION OF FUNCTION: WHOOPS*
WHICH YOU WANT TO CORRECT. ENTER A COMMAND
WHICH DISPLAYS LINE [18] *AND STOPS THE TYPE HEAD AT POSITION* 37:

```
VWHOOPS[18[37]
```

EXCELLENT!

3. *EXAMINE THIS EXPRESSION:*

```
X←(∧/(,ρM)=,ρQ)=(ρM←2 3 4ρι24)∧.=ρQ←3 4 2ρι24
```

WHAT IS THE VALUE OF "X" AFTER EXECUTION?

```
[]:

      1
```

RIGHT!

4. *IF* `V←2 3 4ρ0` *THEN:* `ρV` *IS* 2 3 4.
ENTER THE VALUE OF: `ρρV`

```
[]:
      3
```

YES!

THAT ENDS THE TEST.
EXECUTE: `)SAVE CONTINUE`
SET PAPER TO NEW PAGE,
THEN TYPE: RESULTS

```
      )SAVE CONTINUE
```

15.35.00 10/0 4
/76

RESULTS

APL/360 PROGRAMMING TEST

RESULTS FOR: W . JANSSEN

SCORE (%): 100

ANALYSIS: 10 10 10 10 10 10 10 10 10 10

DATE: 10/04/76
CLOCK: 15:34:43
SIGN-ON: 14:51:43
CONNECT: 00:43:00
CPU: 00:00:02
KEYING: 00:32:56