RM-5367-PR
AUGUST 1967

THE JOSS NOTEBOOK

G. E. Bryan and E. W. Paxson

prepared for
UNITED STATES AIR FORCE PROJECT RAND

Rand
SANTA MONICA, CA. 90406

RM-5367-PR
AUGUST 1967

THE JOSS NOTEBOOK

G. E. Bryan and E. W. Paxson

This research is supported by the United States Air Force under Project Rand—Contract No. F44620-67-C-0045—Monitored by the Directorate of Operational Requirements and Development Plans, Deputy Chief of Staff, Research and Development, Hq USAF. Views or conclusions contained in this study should not be interpreted as representing the official opinion or policy of Rand or of the United States Air Force.

Rand
SANTA MONICA, CA. 90406

Bryan
and
Paxson

THE JOSS NOTEBOOK

RM
5367
PR

JOSS BIBLIOGRAPHY

PUBLICATIONS OF CURRENT INTEREST

Baker, C. L., *JOSS: Introduction to a Helpful Assistant*, The RAND Corporation, RM-5058-PR, July 1966.

———, *JOSS: Console Design*, The RAND Corporation, RM-5218-PR, February 1967.

Bryan, G. E., *JOSS: Accounting and Performance Measurement*, The RAND Corporation, RM-5217-PR, June 1967.

———, *JOSS: Introduction to the System Implementation*, The RAND Corporation, P-3486, December 1966; also published by the Digital Equipment Computer Users Society, *DECUS Proceedings*, Fall 1966.

———, *JOSS: Assembly Listing of the Supervisor*, The RAND Corporation, RM-5437-PR, September 1967.

———, *JOSS: 20,000 Hours at the Console—A Statistical Summary*, The RAND Corporation, RM-5359-PR, August 1967.

———, *JOSS: User Scheduling and Resource Allocation*, The RAND Corporation, RM-5216-PR, January 1967.

———, and J. W. Smith, *JOSS Language: Aperçu and Précis, Pocket Précis, Poster Précis*, The RAND Corporation, RM-5377-PR, August 1967.

Gimble, E. P., *JOSS: Problem Solving for Engineers*, The RAND Corporation, RM-5322-PR, May 1967.

Greenwald, I. D., *JOSS: Arithmetic and Function Evaluation Routines*, The RAND Corporation, RM-5028-PR, September 1966.

[*Continued*]

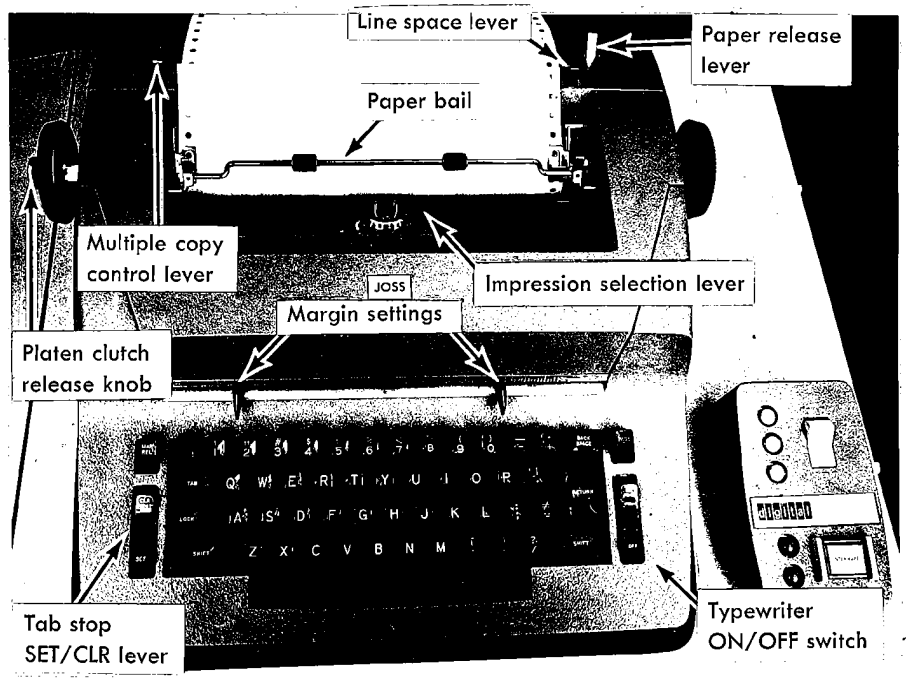
S

T

U

V

W


Line space lever
Paper release lever
Paper bail
Multiple copy control lever
Impression selection lever
Margin settings
Platen clutch release knob
Tab stop SET/CLR lever
Typewriter ON/OFF switch

CONTENTS

INTRODUCTION

JOSS* is an acronym for JOHNNIAC Open-Shop System. JOHNNIAC, now in the Los Angeles County Museum, is the Princeton-type computer designed by John von Neumann and built at The RAND Corporation in 1950–1954. The experimental JOSS system was conceived and implemented on the JOHNNIAC by J. C. Shaw and others. The present system, using a Digital Equipment Corporation PDP-6, was designed by C. L. Baker, G. E. Bryan, I. Greenwald, and J. W. Smith, aided by O. Gross's flaw searches. Currently up to 37 users at typewriter consoles can timeshare the system. (See Fig. 1.) JOSS is now in routine operational status.

This notebook, which is primarily for reference, is designed for someone who has read introductory accounts (see Refs. 1 through 4) and who has experimented with the JOSS console and language. It is a reasonably complete guide to JOSS as currently spoken by the adept, giving by example what he says, and by text why he says it. The machine and program comprising JOSS may be modeled as a single active agent carrying out the user's commands, but we will often speak of a command or verb "doing" something when it is really JOSS that is carrying out the specified action.

The notebook is precisely named. If you had read the large but fragmented JOSS literature, and if almost daily you had hounded the insiders for that sudden illumination of troublesome points, your notes would resemble those that follow. As notes they often ignore or pass over fine points, omitting discussion of specialized matter.

*JOSS is the trademark and service mark of The RAND Corporation for its computer program and services using that program.

[*Continued*]

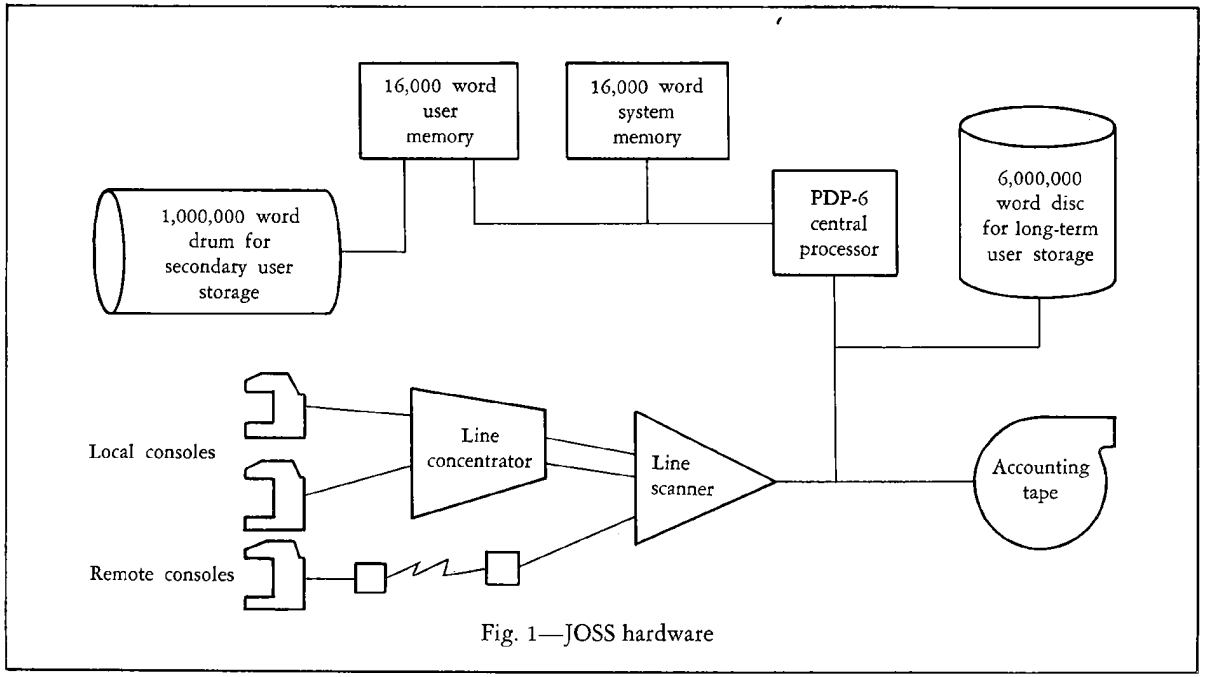


Fig. 1—JOSS hardware

When JOSS first became operational in its present implementation, the language contained a set of features that were thought of as "basic JOSS." Since that time, a few additions have been made that are thought of as "extended JOSS." This notebook does not distinguish between basic and extended JOSS, because it is assumed that the user will be sufficiently skilled in JOSS that it is unnecessary to preserve the distinction.

The pagination follows the JOSS step-number system. Any new sheets replace old ones with the same number and interleave others under the ordering of decimal numbers.

About once a week, the user should:

Use file 108 (RAND8).
Recall item 1.
Do part 1.

for late changes, notes, perhaps a program of the week.

As shown above, all boldface words in this notebook are elements of the JOSS language.

Each point is illustrated by examples from JOSS in which the user speaks in green and JOSS in black. If a question occurs to the reader involving an extension of the illustrated point, he is encouraged to try it on JOSS to see the result.

JOINING THE SYSTEM

- Press POWER ON.
- Respond to housekeeping information demand.
- Press RETURN after each typed input line.
- The top line of each new page (the zeroth line) is administrative—Pacific time, date, the system connection line number, your initials, project number, page number, possibly an administrative message.
- When you hear a series of beeps, press PAGE to see an administrative message.
- When you are in operation and JOSS has control (RED light on), press INTERRUPT to regain control (GREEN light on). If JOSS is typing out, you may have to wait—he wants to be sure you have everything he has completed up to the point of your interrupt.

```
JOSS at your service.
Initials please: GEB
Project number: 1407
Department: CSD
```

Uppercase and/or lowercase letters OK.

1.101

NORMAL TYPEWRITER SETTINGS

● Multiple copy control lever	Middle position
● Line space lever	Forward
● Paper release lever	Back
● Paper bail	Rollers against paper
● Margin settings	At 6 and 84
● Platen clutch release knob	Push in
● Impression selection lever	Set at 3 (middle)

The typewriter ON/OFF switch, located at the right-hand side of the keyboard, will turn the typewriter on and off without connecting the console to JOSS, permitting use as an ordinary typewriter. During JOSS operation, turning this switch off temporarily suspends typewriter activity. It is thus useful when paper needs changing or the user wishes to temporarily quiet output typing. (See photograph of JOSS console, below.)

TABS

Tab stops provide fast mechanical spacing across the page. The mechanical tab stops are set or cleared by pressing the SET/CLR lever at left of keyboard when carrier is at a desired position. In writing forms (see 3.26), you can use tabs instead of tapping the space bar to separate fields. JOSS typeouts will then jump to spread-out positions across the page. (See **Stop**, 3.14, for example.)

INSERTING PAPER

- Obtain a 2- to 3-inch stack of paper. (For paper, call Extension 415.)
- Place wide tear strip with numbers on right.
- Feed paper from box, under platen tray (you use this for backstopping when writing on output), and under rear guide bar and platen.
- After pulling paper release lever to forward position, lift up both small clips over spikes. Push paper under platen and engage spikes at same level on both sides.
- Snap back clips and return paper release lever to normal.

GETTING A NEW PAGE TO START AT TOP

- After logging in, depress PAGE, and wait for typing of page heading.
- Pull platen clutch release knob out.
- Turn platen knobs to get paper fold just visible beyond top of paper bail.
- Push knob in.
- Depress PAGE again to check.

NOTE: Do not use platen knobs to advance paper when operating. JOSS cannot sense this and will fail to give a new page properly. Use RETURN or PAGE.

CHANGING RIBBONS

- Raise typewriter carrier cover.
- There is a lower lever on the right side of the carrier that will raise the ribbon guide clips. Observe how the old ribbon feeds through the clips.
- Lift out cartridge, disengage ribbon, insert new ribbon, seat new cartridge firmly, and depress guide clip lever.
- If you feed the new ribbon through the clips properly, it will not be necessary to touch the ribbon. Ribbon loop may enlarge, but it will tighten during typing.
- For ribbons, call RAND Extension 638.

TROUBLES

Occasionally, console, line, or machine malfunctions may cause the typewriter to "hang up," usually with the RED light on. If this happens, proceed as follows:

- Press INTERRUPT. If signal is getting to JOSS, YELLOW light will come on.
- Rock *typewriter* ON/OFF switch.
- Paper page clutch may be slipping. Noise will alert you. Advance paper using platen knobs.
- Hit carriage return.
- As a last resort, turn power OFF and ON.

TELEPHONE NUMBERS AND MESSAGES

Telephone	*Action*
Extension 233 (EX 3-0437, night)	General information (recorded)
Extension 501 (EX 3-0439, night)	Rings if computer working
Extension 415	Maintenance group

Messages for maintenance group may be left in a JOSS "mail box," File 100 (RAND0). (That's RAND "zero.") Material should be filed without a code in any unused item and written so that **Do part 1.** will display the message. Please include your name, the date, and the time in the message. Responses will be filed in a similar manner, using your initials as a code. Please discard a response once you have read it.

```
Use file 100 (RANDO).
Roger.
Type item-list.
NO ITEMS

1.1 Type "To: Art Lucero  From: EWP 1408 5/14/67".
1.2 Type "Console no. 15, room 1351 fails to space on typeout.".
File all as item 19.
Done.

Delete all.
Recall item 19.
Done.
Type all.

1.1 Type "To: Art Lucero  From: EWP 1408 5/14/67".
1.2 Type "Console no. 15, room 1351 fails to space on typeout.".


Do part 1.
To: Art Lucero  From: EWP 1408 5/14/67
Console no. 15, room 1351 fails to space on typeout.
```

1.141

TELETYPE EQUIVALENCES (Models 33 and 35 TTY)

TTY Keys	TTY Graphic	JOSS Graphic	JOSS Meaning
Control Q	--	--	ON
Control A	--	--	OFF
ALTMODE or ESC	--	--	INTERRUPT
Control L	--	--	PAGE
↑	↑	—	Underscore
Shift L	\	≤	Less than or equal
@	@	≥	Greater than or equal
!	!	|	Absolute value
%	%	≠	Not equal
&	&	·	Multiply (centered dot)
Shift K	[	[	Left bracket
Shift M	]	]	Right bracket
A-Z	A-Z	a-z	Uppercase on TTY, read lowercase by JOSS

NOTE: There is no direct equivalent for backspace; however, RUBOUT types "\" and deletes the preceding character. N RUBOUTs type N "\"s and delete N characters.

```
Type -2*2.
      -2*2 =        -4

Type 1+1/2.
      1+1/2 =        1.5

Type (1+1)/2.
     (1+1)/2 =       1

Type 1/2+1.
      1/2+1 =        1.5

Type 1/3•3.
      1/3•3 =         .999999999

Type 2*3*2.
      2*3*2 =       64

Type 2*(3*2).
     2*(3*2) =      512

Type 2*3•4+5.
     2*3•4+5 =       37
```

-2^2

$1+(1/2)$

$\frac{1+1}{2}$

$(1/2)+1$

$(1/3)\cdot 3$

$(2^3)^2$

2^{3^2}

$(2^3\cdot 4)+5$

2.111

PRECEDENCE RULES

JOSS follows conventional rules in determining the order of operations: exponentiation first, followed by multiplication and division, and then by addition and subtraction. Parentheses, of course, can alter precedence rules, and in fact, in complicated cases, the best advice is to be liberal with parentheses and do a sample side calculation on JOSS, using numbers, to see if you are getting what you want. Many good programs have foundered for lack of attention to precedence.

```
Type 2+10.
        2+10 =        12
type 2+2.                          'Initial capital omitted.
Eh?
Type 2+2                           Final period missing.
Eh?
Type2+2.                           No space.
Eh?
Tpye 2+2.                          Misspelling.
Eh?
Type (2+2•3.                       Unpaired parenthesis.
Eh?
Type 2.2.3.                        Period instead of multiply.
Eh?
Type sin (4.7).                    'Space following function name.
Eh?
Type 2+l0.                         "Ell" instead of "one."
Eh?
Type 2+10.                         "Oh" instead of "zero."
Eh?
Type 2+1 0.                        Space within number.
Eh?
```

2.101

RULES OF FORM

JOSS commands are written as normal English imperative sentences: capitalization of first word, proper spelling and word spacing, final period. There can be only one command per line, which JOSS "considers" only after you punch the carrier RETURN button. A command is executed completely or not at all. In typing mathematical expressions, spaces may be used freely, except within numbers and between the name of any function, formula, or array and the left parenthesis.

EDITING

Line editing must be done before sending to JOSS via carrier RETURN. It can be done even if the final period has been typed. You may backspace, overtype, and use # to blank out unwanted characters. Backspacing or forward spacing does not blank out characters. An asterisk at the beginning or end of a line will tell JOSS upon carrier return to ignore the entire line. The JOSS line may have 78 characters, but it is usually wise to keep lines short, say by abbreviations, to avoid extensive retyping of lines later found to be in error or to require modification.

VALUES

Decimal or logical values may be assigned to any of the 52 uppercase and lowercase letters that JOSS uses for identifiers. Values may be organized into arrays by using indexed letters (2.17). Letters may also be assigned arbitrary expressions called **formulas** by JOSS (see **Let**, 3.18), but a letter may name one value, array, or formula at a time, a new definition replacing the old.

JOSS carries decimal values in the range $\pm 10^{-99}$ to $\pm 9.99999999 \cdot 10^{99}$ with nine digits of significance. Values less than 10^{-99} are replaced by zero.

```
Type 2.0#7 + 5#2.7#1.                    Backspace--overstrike corrections.
2.07 + 52.71 =        54.78

Type#2+2.                                # used to blank out.
          2+2 =         4

Type 2+2 *                               * at end causes line to be ignored.

Type 2+2.
Type 2#+2.                               # indicates transmission error.
Sorry. Say again:
```

2.121

```
1.1 Type "Margins should be set 78 spaces apart. Conventionally at columns 6 and 84.".
Please limit lines to 78 strokes. Say again:
```

```
Set x=3.
Type x•x, x*x.
         x•x =         9
         x*x =        27

Type volts.
Eh?

a(1)=4
a(2)=7
Type a.
        a(1) =         4
        a(2) =        7
Type 10*(-100).
   10*(-100) =         0
Type 10*100.
I have an overflow.

b(1)=true
b(2)=10≤x<100.
Type b.
        b(1) =      true
        b(2) =     false
```

Single letter identifiers only.

Arrays may have 1 to 10 indices.

Logical values too.

```
1.1 Type "a".
1.2 Type "b".
1.3 Type "c".

2.1 Type "d".
2.2 Type "e".

Do part 1.
a
b
c
Do part 2.
d
e

1.25 To step 2.2.

Do part 1.
a
b
e
```

See 3.21 for use of quotes.

Part 1 execution.

Part 2 execution.

Step inserted in part 1.

Note that c and d do not appear.

2.151

ORDER OF PROGRAM EXECUTION

The part is the major unit of program execution. When commanded to do a part, JOSS follows the step sequence within that part, unless directed by a step to go to some place (a step or part) other than the next step. For the verbs that change this order of computation, see **Do**, **To**, **Done**, **Quit**. Otherwise JOSS will *not* move automatically to a next part. A program can be started by executing a part with *any* number. However, most people start with part 1, viewing it as something like an executive routine controlling the whole program.

```
1.1 Type 2+2.                    Stored (indirect) command.
Do step 1.1.                     Direct command.
         2+2 =          4

1.2 Type 3•4.
1.05 Type 5/6.

Type all.

1.05 Type 5/6.                   Steps ordered by step number.
1.1 Type 2+2.
1.2 Type 3•4.

2.1 Type 4*5.
1.2 Type 3•4•5•6.                Replaces previous step 1.2.

Type part 1.
1.05 Type 5/6.                   Steps are stored in order of step number.
1.1 Type 2+2.
1.2 Type 3•4•5•6.

Type part 2.
2.1 Type 4*5.
```

2.141

DIRECT AND STORED COMMANDS

JOSS commands are given in one of two ways. A *direct* command begins with a JOSS verb and is executed immediately (directly) upon carrier return. If the command begins with a number—the step number—it will be stored for later execution. This is the *stored* command, also called the "indirect" command. A sequence of numbered commands forms a stored program in JOSS, to be executed when desired by a direct command. Steps are stored in the sequence given by increasing step numbers (e.g., 1.05, 1.1, 1.19, 1.3), which need not be consecutive. JOSS reorders your input according to these step numbers, regardless of the order of inputting. (Note that 1.19 *precedes* 1.3.)

A sequence of steps with the same number to the left of the decimal is called a **part**, identified by that *integral* number (e.g., **part** 1). The first step of a part may be labeled with an integer, but then it will not be possible subsequently to insert a new step before it. It is good practice to begin with, say, 1.1. A stored step will be deleted and replaced by a new input step with the same number. This automatic replacement is general in JOSS, applying to letters, definitions, and items from files (See 2.19.)

VALUE RANGES

Range of value expressions are used in **for** phrases and in special functions (4.12):

x = a,b,c	means take values a,b, and then c for x.
x = a(b)c	means take values for x from a to c in steps of b.
x = a,b(c)d(e)f,g	means that x will be given successively the values a, b, b+c, b+2c, . . . , d, d+e, d+2e, . . . , f, g.

Note that there is no comma between b(c) and d, since d is the last value for the subrange b(c)d and the first value in the subrange d(e)f. If b+nc<d and b+(n+1)c≥d, the subrange will terminate with . . . , b+nc, d. Then the values d+e, d+2e, . . . , f, and finally g will be used.

```
a=3
1.1 Type i.

Do part 1 for i=1(1)3(a)a*2.
           i =         1
           i =         2
           i =         3
           i =         6
           i =         9

Do part 1 for i=0(1/a)1.
           i =         0
           i =          .333333333
           i =          .666666666
           i =          .999999999
           i =         1
```

Note that expressions may be used.

End of range is always hit exactly.

2.161

ARRAYS

Values may be stored in places named by indexed letters. These arrays may carry up to ten indices whose values must be integers in the closed range -250 to $+250$ (range set by designer's choice in balancing JOSS internal storage layout). A letter may have only one dimension at a time; thus if you have already stored values for a(2,1) and a(3,4) and request that a(2,3,1) be given a value, a(2,1) and a(3,4) will be erased and only a(2,3,1) will remain. Similarly, a nonindexed letter "a" will also be erased. Letters so indexed are used to represent vectors and matrices of any dimension up to 10.

Indexing also plays the role of subscripting, to give greater freedom in naming than that provided by the 52 uppercase and lowercase letters of the alphabet. These uses may be mixed so that the first index stands for a subscript, and those following stand for the array; e.g., $a_i(j,k)$ becomes a(i,j,k) in JOSS. Unlike FORTRAN, JOSS does not *reserve* storage space for arrays according to "dimension" statements, but stores only specified values. (See also 3.25.)

```
Set a(251)=3.
Index value must be integer and |index|≤250.

Set a(1,2,3,4,5,6,7,8,9,8,7)=5.
Please limit number of indices to 10.

Set a(2,1)=5.
Set a(3,4)=10.

Type a(3,4).
      a(3,4) =       10
Type a.
      a(2,1) =        5
      a(3,4) =       10

Set a(2,3,1)=7.
Type a.
    a(2,3,1) =        7

a=3
Type a.
           a =        3
```

Note that entire array is typed.

Old array replaced because of dimension change.

Array replaced by scalar.

2.171

BUGS AND GRONKS

Bugs, those hard-to-find errors that prevent a program from working properly, fall into two major categories. First, the overall flow logic of the program itself may be in error. It is always good practice and discipline to go through the program manually using a few initial values from ranges. In writing this out, the program steps are re-ordered according to the way JOSS will execute them, including iterations. You should also do *sections* of a program on JOSS itself. Second, very subtle errors caused by roundoffs, accumulation of errors, very large values, or values close to zero may arise. You should always know enough about your problem, through previous hand calculation or other means, to detect gross output errors. But for subtle ones, the best advice is, THINK (not a JOSS word) and then consult a numerical analysis man or a Jossmeister.

Gronks are hardware or software failures in the JOSS system. Currently, JOSS is working properly better than 95 percent of scheduled user-hours. If a gronk leads to loss of your current program, JOSS will ask you to log on anew. This may happen, although it rarely does, to an individual user or to all users simultaneously. Periodically filing your program during a session provides protection against these occasional failures.

FILES

The external disc, which holds your files, has a capacity of 45,000 "records," each very roughly equivalent to 60 to 75 JOSS cells. (See 3.23.) The disc can hold 2600 files, each with a maximum of 100 records. At no time can total records in all files in use exceed disc capacity. The file owner can use up to 25 items, with no limitation on item size except that the overall file must not exceed 100 records, called **SPACE** in the *item-list* report. (See 3.28.) If required, one person may own more than one file.

```
1.234567876543  Set x=3.
Please limit step labels to 9 significant digits.

Type 1234.56789876.
Please limit numbers to 9 significant digits.

Set a(2345)=-17.
Index value must be integer and |index|≤250.

Do part 10*99.
Part number must be integer and 1≤part<10*9.

Do step 10*99.
Step number must satisfy 1≤step<10*9.

Type form 5.3.
Form number must be integer and 1≤form<10*9.

Type 10*100.
I have an overflow.
```

Range is $\pm 10^{-99}$ *to* $\pm 9.99999999 \cdot 10^{99}$.

2.181

SUMMARY OF RANGES

The examples below show, by JOSS error messages, the limitations on step, part, form numbers, values, and indices. These ranges may be exceeded inadvertently, since values may be generated by the program.

JOSS OBJECTS

The following words and phrases are called JOSS objects:

step __	**all steps**	**part** __	**all parts**
form __	**all forms**	**formula** __	**all formulas**
all (everything)	**all values** (values of all stored letters)		

These objects may be deleted, typed, or filed. (See **Delete, Type, File.**) The dashes may be filled with a specific number or with an expression that computes a number or formula identifier.

users: This is the number of consoles in use, not just those computing.
time: This is actual Pacific 24-hour clock time.

Note that **users** and **time** appear only in **Type** commands.

ERROR MESSAGES

With one major exception JOSS error messages are on the whole self-explanatory and directly to the point. The exception is **Eh?**. It would be difficult to flag all such errors individually.

Eh? ***Check List*:**

1. Not typed properly. (See 2.10.)
2. Words used not in JOSS vocabulary.
3. Mathematical formatting errors (most frequently a missing multiplication dot or right-hand parenthesis).
4. A space between function, formula, or array name and left parenthesis.
5. Use of "ell" instead of one, "oh" instead of zero, "period" instead of "dot."
6. Attaching a **for** phrase to verb other than **Do**.
7. Typewriter error in sending message to JOSS. Retype.

Some individual error messages of note are the following:

Sorry. Say again.: Retype.
I can't express value in your form.: Usually underscores to left of decimal point are insufficient in number. (An underscore is needed for a minus sign if the value is negative.)
I have too many values for the form.: Check for correct number of fields. Note in the examples, however, that the number of values can be *less* than the number of fields and that underscores can also fill fields.
I have nothing to do.: Calculation in progress is complete or canceled. A new **Do** is needed.

```
Type users.
       users:  19


Type time.
        time:  0921
```

3.101

DO

This is the primary verb that initiates a computation. Upon completion of what the **Do** has ordered, control is returned to the point just after that **Do**. If **Do** is given as a direct command, then control is returned to you. If **Do** is stored, control returns to the step just after the **Do** step in the same part, and computation continues. **Do** can order either a single step or a part to be executed. *Only* the **Do** command can be modified by a **for** phrase. (See 3.19.)

Do is interpreted only once. At that time, all values specified by expressions in the **for** phrase are calculated using current letter values and are saved. Hence, you cannot change the execution range of **Do** by giving new values to the letters in a **for** phrase range, as in FORTRAN. See **Done/Quit** (3.13) for escape methods.

Any **if** clauses modifying **Do** commands are interpreted *only* when the **Do** is *first* met, so that the **if** condition does *not* apply during the subsequent repetitions over the set of values specified by the **for** phrase.

TO

This is a stored command only. It will move computation to the step or part indicated, but will not return control to the step following the **To** step. Computation continues as directed by the program subsequent to the point reached by the **To** command. **To** can send you freely backward and forward in the program.

```
1.1 Type x.

Do part 1 for x=1(1)3 if x≤2.
x = ???
Set x=2.
Do part  1  for x=1(1)3 if x≤2.
           x =        1
           x =        2
           x =        3

1.2 Set x=100.
Do part 1 for x=1(1)3.
           x =        1
           x =        2
           x =        3

Do part A for A=1,2,3.
A = ???
```

The if clause evaluated before Do execution.

The if clause evaluated only once-- before Do execution.

Iteration values saved by Do execution.

Do command interpreted only once.

3.113

```
1.1 Type x.
1.2 Set x=10.
1.3 Type x*2.
1.4 Line.

Do part 1 for x=1(1)4.
          x =        1
        x*2 =      100

          x =        2
        x*2 =      100

          x =        3
        x*2 =      100

          x =        4
        x*2 =      100
```

Note that values of x specified by the for clause are retained and used in successive iterations even though x has a new value set at step 1.2.

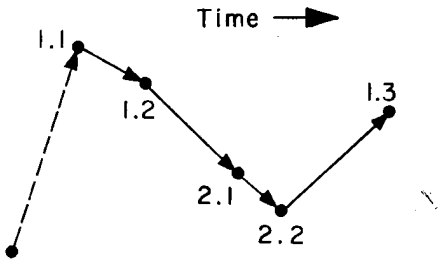
```
1.1 Type "step 1.1".
1.2 Do part 2.
1.3 Type "step 1.3".

2.1 Type "step 2.1".
2.2 Type "step 2.2".

Do part 1.
step 1.1
step 2.1
step 2.2
step 1.3

Do step 2.1.
step 2.1

Do part 2.
step 2.1
step 2.2
```



3.111

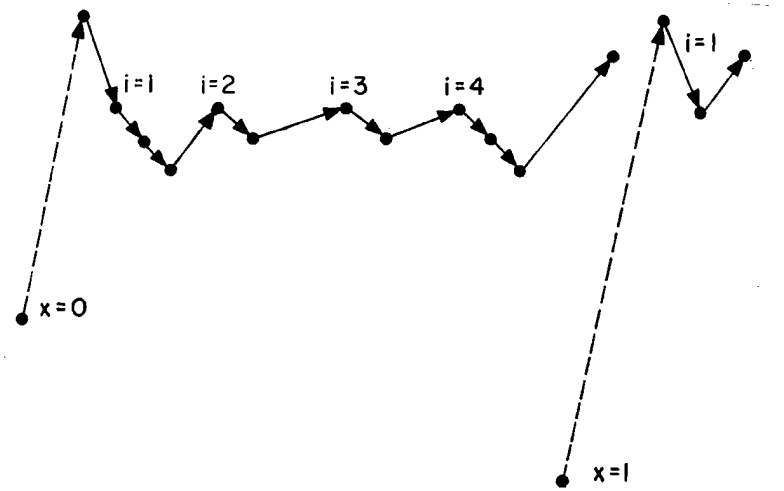
```
1.1 Do part 2 for i=1(1)4.
1.2 Type i.

2.1 Quit if x=1.
2.2 Done if i=2 or i=3.
2.3 Type i in form 1.

Form 1:
*** ***
    —

Do part 1 for x=0.
***1***
***4***
           i =        4

Do part 1 for x=1.
           i =        1
```



3.131

DONE/QUIT

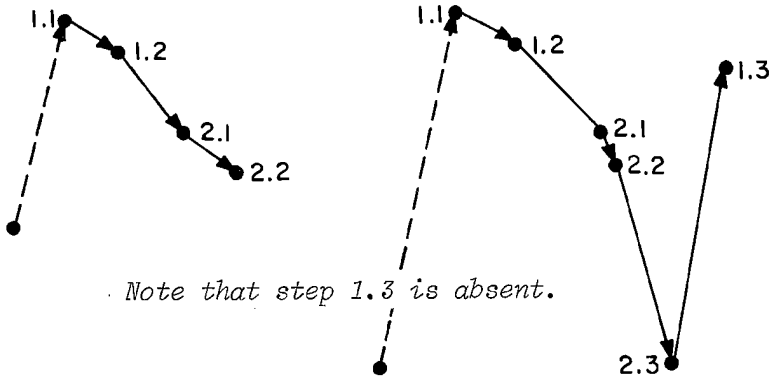
The **Do** command will *normally* be executed for the successive values in the set given by the **for** phrase. If a **Done** statement is reached in this execution, nothing specified *farther on* in the program will be done for the current value being processed. However, the remaining values will be processed. If a **Quit** statement is reached, neither the current value nor any of the remaining values will be further processed. **Done** and **Quit** may have an **if** clause. In this case, the above behavior with respect to the current value and subsequent values is subject to that condition. In JOSS, each part is provided automatically with an implied **Done** command as its last step. That is, as each current value being processed reaches this step, nothing further is done with it. **Quit**, but not **Done**, may be used directly.

```
1.1 Type "step 1.1".
1.2 To part 2.
1.3 Type "step 1.3".

2.1 Type "step 2.1".
2.2 Type "step 2.2".

Do part 1.
step 1.1
step 2.1
step 2.2

2.3 To step 1.3.
Do part 1.
step 1.1
step 2.1
step 2.2
step 1.3
```



Note that step 1.3 is absent.

Compare with Do example 3.111.

3.121

```
1.1 Set i=i+1.
1.2 Quit if i=5.
1.3 Done if i>3.
1.4 Type i.

i=0
Do part 1, 10 times.
          i =         1
          i =         2
          i =         3

Type i.
          i =         5
```

Quit and Done apply also to "times" iterations.

```
1.1 Do part 2 for j=1(1)5.

2.1 Demand a(i,j).
Do part 1 for i=1(1)3.
      a(1,1) = 4                          RETURN pressed, which interrupts.
      a(1,2) =
I'm at step 2.1.

Quit.
Done. I'm ready to go from step 1.1.      Next execution point after 1.1
                                          (the next value for j).

Go.
      a(2,1) =                            RETURN pressed.
I'm at step 2.1.
Quit.
Done. I'm ready to go from step 1.1.      No more action for the Do in 1.1.

Quit.                                     No more action for the direct Do,
                                          so control returns to the user.
Quit.
I have nothing to do.                     There is now nothing to quit.
```

3.133

STOP/CANCEL

Stop is a programmed interrupt (stored only) that returns control from RED to GREEN, back to you. This permits some operator action such as setting tab stops. A subsequent **Go** command will resume computation at the step following the **Stop** step.

Cancel may only be used directly. The machine's knowledge of current position in the computation is erased, without destroying either values computed up to that point or the program itself. A **Do** command is needed to restart the computation. A new **Do** command alone would achieve the same erasure and would start a new computation. But if you are space limited (see 3.23), **Cancel** will give you back maneuver space and let you restart after corrections and insertion of new values. It is a rarely used verb.

```
1.1 Set f(i)=i*2.
Do part 1 for i=-100(1)100.
I'm at step 1.1.                    User presses INTERRUPT.

Cancel.
Go.
I have nothing to do.


1.1 Type "Please set tab stops at columns 15, 25, and 35.".
1.2 Stop.
1.3 Type 15,25,35 in form 1.

Form 1:
        __      __      __          Form contains tabs between fields.

Do part 1.
Please set tab stops at columns 15, 25, and 35.
Stopped by step 1.2.
Go.
        15      25      35
```

3.141

GO

This command continues computation in progress after a programmed or manual interruption, as if the interruption had not occurred. Interruptions may occur because of (1) an error message, (2) pressing INTERRUPT button, or (3) a programmed **Stop** verb. During the interruption, program or values may be modified as desired. If no computation was in progress, although you may have thought so, **Go** will produce the error message of the example. **Go** can only be used after JOSS says he is somewhere.

```
1.1 Type x.
1.2 Stop.
1.3 Type x*3.

Do part 1.
Error at step 1.1:  x  ???
x=2
Go.
          x =         2
Stopped by step 1.2.

1.25 To step 1.25.                    A meaningless loop.
Go.
I'm at step 1.25.                     User presses INTERRUPT.

Delete step 1.25.
Go.
         x*3 =         8
Go.
I have nothing to do.
```

3.151

DELETE

The verb **Delete** followed by one or more JOSS objects, separated by commas, may be used directly or indirectly, and will cause those objects to disappear. Any letter or mixed lists of objects and letters can be erased similarly.

```
Type all.
1.1 Delete part 1.
1.2 Set r=fp(291•r).
1.3 Type "I'm step 1.3".

2.1 Set g=t(100•ip(r)).

Form 1:
____•__     _____     •••••••••

       t(k):  s•f(k)/r(k)

          s =        3.7901

     a(1,1) =        4

Delete step 1.2, part 2, form 1, s, a, formula t.
Type all.

1.1 Delete part 1.
1.3 Type "I'm step 1.3".
Do part 1.
Type all.
```

Steps organized into parts.

Form.

Formula.

Values.

May be used indirectly.

Blank lines follow to indicate nothing stored.

SET

This is a replacement operation that assigns a new *value*—given on the right of the equals sign—to the identifier named on the left (a letter or an indexed letter). **Set** can be direct or stored, but in the direct mode, the word **Set** and the terminal period may be omitted. **Set** computes the right-hand expression just once and stores the value at a given *place*, named on the left-hand side. For indexed letters, values corresponding to each integral index are correspondingly stored in *places* named by the left-hand side. *Note that indices can play the role of subscripts.* (See 3.18 for the important distinction between **Let** and **Set**.) Note that, say, the order **Type A.** will produce all values of A if A is an indexed letter.

```
Let s = sqrt(x*2+y*2).
Type formula s.
          s:  sqrt(x*2+y*2)

Type s.
Error in formula s:  x = ???
x=3
y=7
Type s.
          s =        7.61577311

Let S(x,y) = sqrt(x*2+y*2).
Type S.
      S(x,y):  sqrt(x*2+y*2)
Type S(.1, 10.05).
S(.1, 10.05) =       10.0504975
Type x,y.
          x =        3
          y =        7
Let I(x) = 1≤x<2.
Type I(3), I(1.05).
        I(3) =      false
     I(1.05) =      true
```

Note colon.

Equivalent to Type formula S.

Note that values of x, y are unaffected by Type S.

```
Let f(a,b,c,d,e,f,g,h,i,j,k,l)=a+b+c.
Please limit number of parameters to 10.

Set S(x,y)=sqrt(x*2+y*2).
x = ???

Let S(x,y)=sqrt(x*2+y*2).
Type S(1,2).
      S(1,2) =        2.23606798

Type S(3+45,4•5/17).
S(3+45,4•5/17) =     48.0144153

Let D(f,x)=[f(x+d)-f(x)]/d.
d=.00001


Type D(sin,0), D(cos,0).
    D(sin,0) =        1
    D(cos,0) =        0
```

Any expression may be used for the argument.

Derivative of function f at x. Note that parameters may name formulas, functions, or values.

Derivative of sine and cosine at zero.

LET

This command is used to define functions, called "formulas" in JOSS, of at most 10 variables (called "parameters" in error messages). **Let** defines a rule for computation that is invoked *each* time the function is referenced. Unlike **Set**, the value computed is *not* stored but is used transiently. The "parameters" are dummies—they do not affect the values of stored letters with the same names. (The command **Set f(x)=3.** requires x to be an integer, $|x| \leq 250$, since we are *naming* a storage place, by the letter and the index.)

Let is normally used directly since the defined function is usually employed in the program as a whole. It is used indirectly when in the course of the program the *function is to be redefined* (thus erasing the previous definition).

Let can be used to *abbreviate* an expression (e.g., **Let s=sqrt(x*2+y*2).**). In this case when s is needed, current *stored* values of x and y will be used. If there is no stored x, JOSS will say, **Error in formula s: x=???.** When you type x=1, this becomes a stored value. On the other hand, **Let s(x,y)=sqrt(x*2+y*2).** permits calling via **s(1,2)**. As explained above, x,y are now dummies, and *stored* values are not affected.

```
Set x=x+1.
x = ???
x=3
x=x+1
Type x.
          x =         4
Set a(-1,2)=3.
a(1,-2)=1
Type a.
     a(-1,2) =        3
     a(1,-2) =        1
Set a(1,2,3,-5)=4.
Type a.
 a(1,2,3,-5) =        4

1.1 x=5
1.2 Type x.
Do part 1.
Error at step 1.1:  Eh?
```

The word Set and the period are optional in direct commands.

Array of new dimension replaces old.

The word Set must be used in stored commands.

3.171

```
1.1 Type i.
Do part 1 for i=2(3)10(5)20,100,.003.
          i =         2
          i =         5
          i =         8
          i =        10
          i =        15
          i =        20
          i =       100
          i =          .003
Do part 1 for i=0(1/3)1.
          i =         0
          i =          .333333333
          i =          .666666666
          i =          .999999999
          i =         1
Do part 1, 3 times.
          i =         1
          i =         1
          i =         1
Do part 1, .56 times.
Number-of-times must be integer and ≥ 0.
```

Expressions for numbers could have been used.

Note that range end point is always hit exactly.

Last stored value of i is 1.

3.191

FOR/TIMES

The word **for** gives the set of values (see 2.16) for which a **Do** order is to be executed. It can be used *only* with the verb **Do**. The set of values in the **for** phrase is stored, as if a **Set** command had been used for each value successively.

An additional modifier for the **Do** command only is **times**:

Do step __, n times.
Do part __, n times.

Note the comma, which makes it possible to avoid an unpleasant conjunction of numbers, and the integer n, which, *as always in JOSS*, may arise from a computation.

See 4.16 for Conditional

```
Let B(i) = [ i=0:L; B(i-1)<P:0; B(i-1)•(1+r)-P ].
L=1000
P=100
r=.005

1,1 Type I,B(I) in form 1.
Form 1:

___  ____.__
Do part 1 for I=0(1)11.
  0    1000.00
  1     905.00
  2     809.53
  3     713.57
  4     617.14
  5     520.23
  6     422.83
  7     324.94
  8     226.57
  9     127.70
 10      28.34
 11        .00
```

Formula B calculates, in a recursive way, the balance of loan L which has payment P and interest rate r.

IF

This word may be used to qualify *any* command. It specifies the conditions under which the imperative will be executed. These conditions may contain logical and algebraic expressions. In examining a command with an **if** clause, JOSS first evaluates the condition in that clause. When the condition is not met, the command is not processed in any way. Thus the command could be in error and the error not detected until the condition is finally met. (See also examples under **Do**, 3.11.)

TIMER

This measures real time in minutes and hundredths of minutes since the instant of console activation *or* since the last (unique) command **Reset timer.**. The word **timer** may be used in computational expression (hence minutes and hundredths). (See 5.13–5.15 for some uses of **timer**.)

```
x=3
Let y=x*2.
1.1 Type x,y in form 3.
Form 3: -
x=__   y=__    appear in form 3.
Type x, step 1.1, form 3, formula y.
          x =        3
1.1 Type x,y in form 3.
x=__   y=__    appear in form 3.
          y:  x*2

Type "JOSS".
JOSS
Type ""JOSS"JOSS"".
"JOSS"JOSS"

Type x,__,y.
          x =        3

          y =       9
```

TYPE

Type may be direct or stored. Any JOSS object (3.10) can follow **Type**, as may any letter or expression, or any list of objects, letters, and expressions separated by commas. The command causes appropriate typeout on successive lines.

If double quotes are placed around any expression or text, it will be typed out verbatim *without quotes*. (Single quotes are not part of JOSS punctuation.) The underscore will cause a corresponding blank *line* in the typeout.

The JOSS word **in** appears *only* in the command **Type** __ **in form** __.. (See 3.26.)

```
x=1
y=4
Type x if x=1 and y<34.
           x =        1

Type x if x=3 and not (x+y>34) or y≠17.
           x =        1
Set y=35 if x<40.
Type y.
           y =       35

1.1 Type 1≤x<3<y≤a.
Do step 1.1 for a=15,100.
   1≤x<3<y≤a =      false
   1≤x<3<y≤a =      true

Do step 1.1 if y>10*6.

Type garbage if y=0.
```

Compound condition.

Condition fails, so command is not interpreted

Condition fails, so command is not interpreted.

3.201

SIZE

An individual JOSS user has at his disposal a maximum of about 1900 cells (storage areas), even if he is the sole user. A cell will hold one numerical value (but some overhead cells are required for array rows and columns) or six characters of a program step, form, or formula. The response to the command **Type size.** is the number of cells currently in use. Since **size** may be used in computational expressions, the user may arrange his program to alert him to an imminent out-of-space error message. During the execution of a **Do** command, cells are also required to record the progress of that execution. Similarly, cells are needed for the evaluation of each formula. (For extraordinary cases, where more size is needed, consult the JOSS team. Note that files (2.19) and chaining (3.30 and 5.18) may be used for large problems.)

```
Type timer.
      timer =       110.16

Reset timer.

Type timer
Set x=timer.
Type timer-x.
     timer-x =          .2

Set timer =5.
Eh?
```

Value of timer may not be set.

3.221

```
Type size.
        size =         0
1.1 Set a(i,j)=i+j.

2.1 Do part 1 for j=-250(1)250.

Do part 2 for i=-250(1)250.
I'm at step 1.1. I ran out of space.

Type size.
        size =      1895

Cancel.
Type size.
        size =      1879

Delete a.
Type size.
        size =        14
```

Cancel retrieves 16 cells that were used for the execution records of Do.

3.231

DEMAND

This is a stored command only. The order allows the program to call for values to be typed in by the user, thus minimizing the amount of typing to be done, systematizing inputs, and avoiding failures to input needed data. JOSS types out the requested letter, which may be indexed, and an equals sign. The user types the desired value or expression to the right. If the user terminates his input line with an asterisk, the line is ignored and the **Demand** request is made again. If the user's response is improperly formed, JOSS returns an appropriate error message and makes the request again. If the user hits RETURN only, JOSS considers it to be an interrupt and returns control to the user with an appropriate message.

If the **Demand** command is followed by the word **as** and a string of text in quotes, JOSS will type the quoted text followed by an equals sign to identify the requested value.

FORM

A **form** describes how you want your output to appear. In the body of the program, the order is: **Type** "list" **in form n.**. The "list" is usually a set of letters or expressions, separated by commas, whose values will be typed out following the form's layout. The form number n is an integer ($1 \leq n < 10^9$) but may be an expression whose value will be computed.

Forms are typed directly. The first line is simply **Form n:**. The second line is a set of fields and/or internal information, spaced as desired.

A field is a string of underscores, perhaps with a decimal point, or a string of periods. Values will fill these fields in the order specified, rounded as necessary. *All other characters in the form are typed out exactly as they were typed in* (including asterisk).

The number of underscores to the right of a decimal point indicates the desired roundoff. The number of underscores to the left *must* be adequate to hold the expected integer part of computed values, including a minus sign. It is usually simplest to be generous. A field of periods will be filled by the value in scientific notation. The *minimum* field of 7 periods will give *just* two significant digits. The first period is reserved for the sign of the value, and the third from the end is reserved for the sign of the exponent.

Forms without fields are useful to output headings, messages, and the like. In this case, the second line of the form definition is the desired text and the command is simply **Type form n.**.

```
a(2,3,4)=7
Type a(2,3,5).
a(2,3,5) = ???
Let a be sparse.
Type a(2,3,5).
    a(2,3,5) =        0
Type a.
    a(2,3,4) =        7
a is sparse

a(2,3)=23
Type a.
      a(2,3) =       23
Type a(1,2).
a(1,2) = ???
Let a be sparse.
Type a(1,2).
      a(1,2) =        0
```

Because the command requested all values of a.

New array replaces old.

3.251

SPARSE

This term is used only in the command **Let A be sparse.**, where A is the name of an array with up to 10 indices. When A is declared sparse and values have been given to some of the array's elements, JOSS considers all other elements to be zero. If a new A is entered with a different number of indices, the old A is erased and the new A must be redeclared sparse. Inputting elements is a **Set** operation, and hence only these elements are stored. The command corresponds to **Let** in that nonstored elements are "computed" to be zero. Hence **sparse** saves storage space and inputting effort.

```
1.1 Demand a(i).

Do part 1 for i=1(1)5.
a(1) = 10
a(2) = 3.5+24
a(3) = sin(4.789)
a(4) = a(2)+3•a(3)
a(5) = 28.75*
a(5) = atemp
Eh?
a(5) =
I'm at step 1.1.
a(1)=100

Go.
a(5) = 15

1.1 Demand E as "Voltage maximum".

Do part 1.
Voltage maximum = 135.5
```

Any expression may be given.

Input ignored and request repeated.

Carrier return only is an interrupt.

```
1.1 Type form 2.
1.2 Line.
1.3 Do part 2 for x=10(10)60.
2.1 Type sin(x), log(x), exp(x) in form 1.
Form 1:

___._____     ___._____     ........
Form 2:
SIN(X)        LOG(X)        EXP(X)

Do part 1.
SIN(X)        LOG(X)        EXP(X)

  -.54          2.30          2.20 04

   .91          3.00          4.85 08
  -.99          3.40          1.07 13

   .75          3.69          2.35 17
  -.26          3.91          5.18 21

  -.30          4.09          1.14 26
```

See 3.27 for Line.

Form with literal information only.

```
Form 1:
___._ amps.  ___._ volts.
Form 2:
___._ amps.  ..... volts (voltage ≥ 1000).

1.1 Type I, I•R in form (I•R<1000:1; 2 ).
R=91

Do step 1.1 for I = 8.735(1.05)12.9.
  8.7 amps.  794.9 volts.
  9.8 amps.  890.4 volts.
 10.8 amps.  986.0 volts.
 11.9 amps.   1.1 03 volts (voltage ≥ 1000).
 12.9 amps.   1.2 03 volts (voltage ≥ 1000).
```

Form number is computed (see 4.16).

Form 1 used.

Form 2 used.

3.262

```
Form 1:
____    ___._    .......

x=12.356
Type x,x,x in form 1.
  12     12.4     1.2 01

Type x,x,x,x in form 1.
I have too many values for the form.

Type x,x in form 1.
  12     12.4

x=1234.5678
Type x,x,x in form 1.
I can't express value in your form.

Form 4:
........................
Type -2/3 in form 4.
-6.666666700000000000-01
```

Note rounding into form.

Fewer values than fields OK.

Note rounding to 9 significant digits.

3.261

LINE/PAGE/$

Line. advances paper one line, leaving a blank. (See also 3.21.) **Page.** advances paper to start a new page. These commands can be used to improve output format. The format can be programmed by appending **if** clauses to these verbs. The symbol $ has as value the current line number on the page (1 to 54—the heading line is for administrative use only). The symbol $ is often used in **if** clauses for format control purposes.

FILE/DISCARD/RECALL

All file commands may be direct or stored.

The command to *enter* material in your file is **File** "list" **as item n (code).**, where the "list" consists of JOSS objects (3.10) and/or letters identifying values, arrays, or formulas. You choose n and, if you wish, an identifying code of up to five letters or numbers, frequently a mnemonic.

To erase an item from your file, freeing that number, the command is **Discard item n (code).**.

To bring a *copy* of an item from the files, the command is **Recall item n (code).**. Since it is a copy, you can manipulate or modify the item as you wish. To replace the original item with a modification, you first **Discard** and then **File.** After **Recall**, you can use **Type** commands to examine the item.

In all cases, a space must appear between **n** and **(code)**.

```
Use file  107 (rand 7).
Please limit ID's to 5 letters and/or digits.
Use file 107 (rand7).
Roger.
Type item-list.
ITEM  CODE    RPN      DATE   SPACE
 1    EX1     1407     5/07/67     1
 3    EX3     1407     4/12/67     1
 4    EX4     1407     4/12/67     1

Discard item 2.
Recall item 3 (ex3).
Done.
File all as item 1.
Please discard the item or use a new item number.
```

Space counted.
Case of ID letters immaterial.

USE/ITEM-LIST

The command **Use** *initiates* all file actions, but needs to be used only once for each different file. The complete command is **Use file N (ID).**. Here N is your assigned file number. The identifier (ID), also assigned, is a code of no more than five letters or numbers. It does not matter whether the letters in the code are uppercase or lowercase. In all cases, a space must appear between **N** and **(ID)**.

Contents of your file may be reviewed by the command **Type item-list.** (note hyphen). In this printout, the heading **SPACE** refers to the number of disc records (see 2.19) required to store the item. **DATE** refers to the time that the item was filed, and **RPN** is the RAND project number.

```
Page.

15:06  7/31/67   #24  geb  1407      [3]


Type $.
          $ =        2
Line.

Type $.
          $ =        6
1.1 Line if fp($/3)=0.
1.2 Type $.
Do part 1, 4 times.
          $ =       10
          $ =       11
                                    Line command executed.
          $ =       13
          $ =       14
```

```
Type all .

1.05 Set r=fp(r•291).

2.45 Do part 17 for x=.003, .07(.03).15.
Recall item 16 (ex38).
Done.
Type all.

1.05 Set r=fp(r•291).
1.2 Do part 10 if $≠1.
1.3 Do part 2 for i=1(1)10.

2.2 Set b=-a.
2.3 Type b,f(b),log(f(b)),I(f(b)) in form 1.
2.45 Do part 17 for x=.003, .07(.03).15.
```

Recalling from the files results in a mixture of that on hand and that coming from the files.

3.301

```
b(1) =        3
b(3) =        7
```

CHAINING

It is also possible to "chain" subprograms or data held in separate items and needed for a large problem. You will not run out of space or mix part numbers provided sufficient care is taken in programming the chain's housekeeping. (See 5.18.)

FILE WARNING

Recalling file material is equivalent to typing in the filed material from a console during the session. Hence step numbers, values, etc., from the item replace similarly numbered or named ones previously typed in. It is good practice to **Delete all.** before recalling an item. Otherwise, rather appalling mishmashes of steps can result. Manipulations of files and items can be done indirectly, by program, and this is most useful.

File access time may vary from a few seconds to minutes, depending essentially on the number of users queueing for file access. An impatient interrupt will send you back to the queue's end.

```
1.1 Page.
1.2 Type form 2.
1.3 Do part 2 for i=1(1)10.

2.3 Type b,f(b),log(f(b)),I(f(b)) in form 1.

File all as item 16 (ex38).
Done.
Delete all.
Recall item 16 (ex38).
Done.
Delete step 1.1.
1.2 Do part 10 if $≠1.
Discard item 16 (ex38).
Done.
File all as item 16 (ex38).
Done.
Type all.

1.2 Do part 10 if $≠1.
1.3 Do part 2 for i=1(1)10.

2.3 Type b,f(b),log(f(b)),I(f(b)) in form 1.
```

Copies all steps, forms, formulas, and values into the file.

Copies from file to computer.

Program is modified. Must discard old version before filing updated one.

3.291

PARENTHETIC COMPUTATION

Sometimes in the course of a calculation (say, at an interrupt point or an error message) it is convenient to perform a second calculation (perhaps to identify an error) without disturbing the one in progress. This may be accomplished by a parenthetic **Do**—a **Do** command enclosed in parentheses. The process may be carried out indefinitely (subject to storage limits). Parenthetic **Cancel.** cancels the current subexecution, and the primary execution may then be resumed. Note that new values so computed for letters will remain after the parenthetic computation.

NUMBER DISSECTION FUNCTIONS

Signum:	**sgn(x)** $= -1,0,+1$	as $x<0$, $x=0$, $x>0$,
Integer part:	**ip(x)**	carries the sign of x,
Fraction part:	**fp(x)**	carries the sign of x,
Digit part:	**dp(x)**	carries the sign of x,
Exponent part:	**xp(x)**	carries the sign of the power of 10.

Note that

$$x = \mathbf{ip}(x) + \mathbf{fp}(x),$$

$$x = \mathbf{dp}(x) \cdot 10^{\mathbf{xp}(x)}.$$

Absolute value: $|\ldots|$.

```
Type sqrt(2), log(10), exp(1), sin(3.1415).
     sqrt(2) =        1.41421356
     log(10) =        2.30258509
      exp(1) =        2.71828183
 sin(3.1415) =        9.26536•10*(-5)

Type exp(log(13.1)).
exp(log(13.1)) =     13.1

c=1/log(10)
Let L(x) = c•log(x).                                    Log base 10.
Let T(x) = arg(1,x).                                    Arctangent.
Let S(x) = arg(sqrt[1-x*2],x).                          Arcsine.

Type L(10), L(100), T(0), T(10000), S(1), S(0).
       L(10) =         .999999999
      L(100) =        2
        T(0) =        0
    T(10000) =        1.57069633
        S(1) =        1.57079633
        S(0) =        0
```

4.101

BASIC FUNCTIONS

In typing *any* function, there must be *no* space between the function name and the left parenthesis or bracket.

Square root:	**sqrt(x)**	$x \geq 0$,
Natural log:	**log(x)**	$x > 0$,
Exponential:	**exp(x)**	$e^x < 10^{100}$ (use exp(1) for e),
Sine, cosine:	**sin(x),cos(x)**	$\|x\| < 100$, radian measure.

JOSS computes the true value rounded to nine significant digits in most cases. Care is also taken to hit certain "magic values" (familiar argument values) exactly; i.e., computational algorithms are modified at these values.

Argument:	**arg(x,y)** is the angle of the point x,y in radians; **arg(0,0)** = 0 by definition; **arg(−1,0)** = π.
Arcsine:	$\sin^{-1}(x)$ = **arg[sqrt(1−x*2),x]**.
Arc-cosine:	$\cos^{-1}(x)$ = **arg[x,sqrt(1−x*2)]**.
Arctan:	$\tan^{-1}(x)$ = **arg[1,x]**.

See 5.16 for degree–radian conversion.

```
1.1 Demand a(i,j).
2.1 Do part 1 for i=1(1)5.
Do part 2 for j=1(1)5.
      a(1,1) = 1
      a(2,1) = 2
      a(3,1) = 3
      a(4,1) = 4
      a(5,1) =                        Carrier return interrupts Demand.
I'm at step 1.1.
(Do part 2 for j=1.)
      a(1,1) = 2                      Parenthetic Do execution used to
      a(2,1) = 3                      reenter some values.
      a(3,1) = 4
      a(4,1) =
I'm at step 1.1.
(Cancel.)                             Parenthetic execution is canceled.
Done. I'm ready to go at step 1.1.
Type i.
          i =        4
i=5
Go.                                   Correct value of i set and suspended
      a(5,1) = 5                      execution resumed.
      a(1,2) = 4+2
      a(2,2) = 3+2
      a(3,2) =
```

3.311

```
x = 100.7

Type min[ x=.1(.1)4: x*2-2•x ].
min[ x=.1(.1)4: x*2-2•x ] = -1

Type x.
          x =      100.7

Type max(1,2,3,4,5,1/2,sin(5),.1,log(x)*2,.01).
max(1,2,3,4,5,1/2,sin(5),.1,log(x)*2,.01) = 21.2718889

Type min[ x=1(.1)10: (x-1.5)•(x-9.3) ].
min[ x=1(.1)10: (x-1.5)•(x-9.3) ] = -15.21
```

The x within the min is a dummy.

4.121

SPECIAL FUNCTIONS

In defining the special JOSS functions, the notation x = a(b)c:f(x) means that f(x), any function or expression involving x and including the constant function, is to be evaluated for x = a,a+b,a+2b, . . . ,c, where a,b,c need not be integers (indeed they can be any expressions), and it is not necessary that a+nb = c (see 2.16). The letter x is a dummy variable in the sense that the value of this same letter, if already stored (say by **Set**) will not be affected.

Maximum: **max(a,b,c,d)** or **max(x = a(b)c:f(x))**.
(Gives the largest in the list a,b,c,d or the largest f(x) in the range a(b)c.)

Minimum: **min(a,b,c,d)** or **min(x = a(b)c:f(x))**.

Sum: **sum(a,b,c,d)** or **sum(x = a(b)c:f(x))** means a+b+c+d or f(a)+f(a+b)+ . . . +f(c), and, in particular,

$$\sum_{i=1}^{N} f(i) = \textbf{sum(i = 1(1)N:f(i))}.$$

[*Continued*]

```
Set x = -123.4567.

Type x,sgn(x),ip(x),fp(x),dp(x),xp(x),|x|.
           x =     -123.4567
      sgn(x) =       -1
       ip(x) =     -123
       fp(x) =        -.4567
       dp(x) =       -1.234567
       xp(x) =        2
         |x| =      123.4567

Type x=ip(x)+fp(x), x=dp(x)•10*xp(x).
x=ip(x)+fp(x) =      true
x=dp(x)•10*xp(x) =      true
```

Product: **prod(a,b,c,d)** or **prod(x = a(b)c:f(x))** means a·b·c·d or f(a)·f(a+b) . . . f(c), and, in particular,

$$\prod_{i=1}^{N} f(i) = \textbf{prod(i = 1(1)N:f(i))}.$$

First: **first(x = a(b)c:** any mathematical or logical condition involving x**).**

(Gives the first, and *only the first*, value of x in the range, for which the condition is satisfied.)

TRANSLATION VALUE

JOSS makes a careful distinction between decimal values and logical values in its internal storage. It carries this distinction to the user by using the words **true** and **false** when a letter or expression has a logical value. Since it is common practice to use the decimal value 1 for true and 0 for false, JOSS provides conversion between the two value forms by the translation value function **tv(x)**:

tv(P) = 1	for **P = true**, P a satisfied condition,
tv(P) = 0	for **P = false**, P not satisfied,
tv(x) = true	for **x ≠ 0**, x a number,
tv(x) = false	for **x = 0**.

We also have

|P| = 1	for **P = true**,
|P| = 0	for **P = false**.

```
1.1 Type a  or  b and not  c  or  d.
1.2 Type (a or b) and not (c or  d).
b=false
c=true
d=false
Do part  1 for a= true, false.
a  or  b and not  c  or  d =      true
(a or b) and not (c or  d) =      false
a  or  b and not  c  or  d =      false
(a or b) and not (c or  d) =      false

Type 1=1, 2<.03 and 5>25 or not 17≥17.
         1=1 =      true
2<.03 and 5>25 or not 17≥17 =      false

Type false=false=true.
false=false=true =      false
Type (false=false)=true.
(false=false)=true =      true
```

4.131

LOGICAL FUNCTIONS (EXPRESSIONS)

The words **not**, **and**, **or** (inclusive) can be used freely to form functions or expressions and are used extensively in conditions (**if** clauses, conditional expressions, **first** function). These words are used with letters or propositions having a truth value and not with letters having numerical values. Given a mathematical expression or logical proposition P that is either true or false, JOSS can be asked to **Type P.**. JOSS will respond **P = true** or **P = false.**

Precedence is (1) **not**, (2) **and**, (3) **or** in evaluating logical expressions. JOSS would interpret "A or B and not C or D" as "A or (B and (not C)) or D." It seems wise to make your desires clear by using parentheses in constructing the expression.

```
Type sum(1,2,3,4,5).
sum(1,2,3,4,5) =     15
Type sum(i=1(1)10: i*2 ).
sum(i=1(1)10: i*2 ) = 385

Type prod(x=1(1)6:x).
prod(x=1(1)6:x) =   720

Type first(x=1,2,3,4,3.5,4.8,7+8,-1*3: x>2).
first(x=1,2,3,4,3.5,4.8,7+8,-1*3: x>2) = 3

Type min[ x=1(.1)10: (x-1.5)·(x-9.3) ].
min[ x=1(.1)10: (x-1.5)·(x-9.3) ] = -15.21

Type first[ x=1(.1)10: (x-1.5)·(x-9.3) = -15.21 ].
first[ x=1(.1)10: (x-1.5)·(x-9.3) = -15.21 ] = 5.4

Type first(x=1,2,3,4,5: x=2.5 ).
first(x=1,2,3,4,5: x=2.5 ) = ???
```

Factorial.

4.123

```
a=3
b=5
Type conj[ 4>a≠10, a•b≥a+b, b>4 or a>4 ].
conj[ 4>a≠10, a•b≥a+b, b>4 or a>4 ] =      true

Type disj[ a-b>0, b+3•a=7, a*2•b≥25 ].
disj[ a-b>0, b+3•a=7, a*2•b≥25 ] =      true

1.1 Type conj[ i=1(1)b: a<i≤c ].
1.2 Type disj( x=1(1)b: a≤x≤c ).

Do part 1 for c=4,1.
conj[ i=1(1)b: a<i≤c ] =      false
disj( x=1(1)b: a≤x≤c ) =      true
conj[ i=1(1)b: a<i≤c ] =      false
disj( x=1(1)b: a≤x≤c ) =      false
```

List of arguments.

Range of values.

CONJUNCTION/DISJUNCTION

The function **conj(i = 1(1)n:P(i))** = **true**, if P(1) and P(2) and . . . P(n) are all **true**. Otherwise the function **conj(i = 1(1)n:P(i))** = **false**.

The function **disj(i = 1(1)n:P(i))** = **true**, if at least one of P(1),P(2), . . . ,P(n) is **true**; otherwise it is **false.**

```
Set x=true.                                    Value true stored at x.
Type tv(x).
       tv(x) =         1
1.1 Type tv(x),|x|.
Do step 1.1 for x=5, 0, -.3, true, false.
       tv(x) =      true
         |x| =         5
       tv(x) =     false
         |x| =         0
       tv(x) =      true
         |x| =         .3
       tv(x) =         1
         |x| =         1
       tv(x) =         0
         |x| =         0
Type tv(2>3).                                  Transient calculation.
     tv(2>3) =         0
Set p = 2>3.
Type p,tv(p),2>3.
           p =     false                       Value false stored at place p.
       tv(p) =         0
         2>3 =     false
```

CONDITIONAL FUNCTIONS (EXPRESSIONS)

The notation **(A:b;C:d;e)** is read as follows:

If the mathematical condition/logical proposition A is satisfied/true, then use the value or expression b. If A is **false** and C is **true**, use d. If A and C are **false**, use e (which need not be preceded by a condition). Length of string is limited only by line capacity. Reading from left to right, the first true condition will determine the expression used. Conditional expressions can be used anywhere.

Functions can be defined freely by

Let f(x) = (A:b(x);C:d(x);e)..

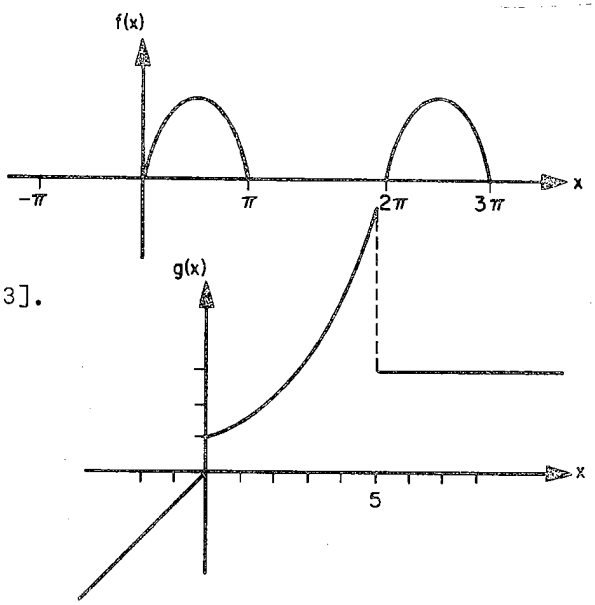
```
Type [1=2:1;2] + (1≠2:3;4).
   [2] + (3) =        5

Let f(x) = (sgn(sin[x])=-1:0; sin(x)).
Type f(-1), f(2), f(4).
       f(-1) =        0
        f(2) =         .909297427
        f(4) =        0

Let g(x) = [x<0:x; 0<x<5:exp(x/2.5); x≥5:3].
1.1 Type ẋ, g(x) in form 1.
Form 1:

__    ___.__
Do part 1 for x=-1(2)3,4,7.
-1     -1.00
 1      1.49
 3      3.32
 4      4.95
 7      3.00

Do part 1 for x=0.
Error at step 1.1 (in formula g):  [x<0:x; 0<x<5:exp(x/2.5); x≥5:3] = ???
```



Note that g(x) is undefined at x=0.

4.161

RECURSIVE FUNCTIONS

Speaking loosely, a function is defined recursively if it appears in its own definition. If f(x) is the function to be defined, and g(x,y) is a given function, then f(x) is defined recursively by f(x) = g(x,f(x−1)).

Each time a function is evaluated by JOSS, space is required to hold temporarily the progress of the execution *including* partial results needed to complete the calculation.

If we say **Let f(x) = x·f(x−1).**, and then **Type f(2).**, JOSS will try f(2) = 2·f(1) = 2·1·f(0) = 2·1·0·f(−1) = ... JOSS will *not* stop when 2·1·0·f(−1) is reached because the computation, in his view, is not completed. Since space is needed for these partial results, JOSS types **I ran out of space.**. Some condition is always needed in conjunction with a recursive definition.

```
Let f(x) = x•f(x-1).
Type f(3).
Revoked. I ran out of space (in formula f).

Let f(x) = [ x=0:1; x•f(x-1) ].      If x=0, f(x)=1; otherwise x•f(x-1).
Type f(6).
       f(6) =      720
Type f(0).
       f(0) =        1

Let f(x) = ( x=0:1; fp(x)=0:prod(i=1(1)x:i) ).    f(x)=x! (factorial).
Type f(0), f(6).
       f(0) =        1
       f(6) =      720

Type f(5.4).
Error in formula f:  ( x=0:1; fp(x)=0:prod(i=1(1)x:i) ) = ???
```

4.171

PROGRAM LAYOUT

Most JOSS users are "Topsy" programmers: the program gets lashed up at the console—which, of course, is the idea behind JOSS. But for *large* problems it is an inefficient use of man and machine time, and ties up a scarce resource—consoles.

Large problems should be *blocked out* first at the desk. Define the main task, the subtasks generated, etc., to get the problem "tree." Indicate which subtasks must precede which others. List formulas and abbreviations. View each JOSS part as being responsible for executing a subtask. Part 1 may be the overall executive. Establish input and output housekeeping formats.

```
Do part 1.
I can compute the volume of a sphere(1), cylinder(2), or cone(3)
if you will indicate which, please.
          w = 3
Radius?
          r = 3.1
Height?
          h = 5.0
The volume is  50.32
          w = 1
Radius?
          r = 34.057
The volume is  1.65 05
          w = 2
Radius?
          r = 800
Height?
          h = 10*6
The volume is  2.01 12
          w = 5
How was that again?
          w =
```

5.111

VOLUME CALCULATION

```
Delete all.
Recall item  5 (ex5).
Done.
Type all.

1.1 Type "I can compute the volume of a sphere(1), cylinder(2), or cone(3)".
1.2 Type "if you will indicate which, please.".
1.3 Demand w.
1.35 Type "How was that again?" if t.
1.36 To step 1.3 if t.
1.4 Do part 2.
1.5 Type v in form  (v<1000:1;2).
1.6 To step 1.3.

2.1 Type "Radius?".
2.2 Demand r.
2.3 Done if w=1.
2.4 Type "Height?".
2.5 Demand h.

Form 1:
The volume is ___.__

Form 2:
The volume is ........

          t:  not (w=1 or w=2 or w=3)
          v:  (w=1:4/3•p•r*3;w=2:p•r*2•h;w=3:p/3•r*2•h)

          p =         3.14159265
```

5.11

PRIME NUMBERS

```
Delete all.
Recall item 1 (ex1).
Done.
Type all.

1 Type x if P(x).

2 Type x in form 1+tv(P(x)) if (x=true or x= false:true;fp(x)=0).

Form 1:
_____ is not prime.

Form 2:
_____ is prime.

      P(x):  (x=true or x=false:false;x≤1 or fp(x)≠0:false;p(x))
      p(x):  first[i=2,3(2)[x<10:3;ip(sqrt(x))],x:fp(x/i)=0] =x

         x =       54
```

The function P(x) has value true if and only if x is prime. After P(x) filters out invalid cases its subfunction p(x) finds the first exact divisor of x--via fp(x/i)=0 and compares that divisor with x. Then y is prime if the first exact divisor is x itself.

5.12

```
Do part 2 for x=1, 2, -5, .03, true, false, 17.
          1 is not prime.
          2 is prime.
         -5 is not prime.
       true is not prime.
      false is not prime.
         17 is prime.
Do part 1 for x=1(1)30.
          x =        2
          x =        3
          x =        5
          x =        7
          x =       11
          x =       13
          x =       17
          x =       19
          x =       23
          x =       29
```

5.121

COMPUTING EFFICIENCY—I

```
Type all.

1.1 Set m=M.
1.2 Set g=first(x=0(a)10:f(x)=m).
1.3 Type m,g.

2.1 Type M,G.

4.1 Line.
4.2 Reset timer.
4.3 Do part k.
4.4 Type k,timer in form 1.

Form 1:
Minutes for case _:  __.__

Let G = first(x=0(a)10:f(x)=M).
Let M = min(x=0(a)10:f(x)).
Let f(x) = x*3-10•x*2-6•x+10.
```

This example gives some idea of economy in computing the first value of a function where a minimum is achieved. Part 1 (k=1) computes the minimum once and stores it. Part 2 (k=2) computes the minimum anew for each value of x. Here timer shows only relative efficiency since elapsed time depends on the number of users computing.

```
a=.1
Do part 4 for k=1,2.

          m =     -179
          g =        7
Minutes for case 1:    .03

          M =     -179
          G =        7
Minutes for case 2:   1.46

Type users.
      users:  9
```

Note that if we Let G(z)=first (x=0(a)10:f(x)=z). and Type G(M), then M will only be calculated once.

COMPUTING EFFICIENCY—II

```
Delete all.
Recall item 10 (tbase).
Done.
Type all.

4.1 Line.
4.2 Reset timer.
4.3 Do part k.
4.4 Type k,timer in form 1.

Form 1:
Minutes for case _:  __.__

1.1 Do part 10 for x=-200(1)200.
10.1 Set d=-f(x-1)+f(x).
2.1 Do part 20 for x=-200(1)200.
20.1 Set D=f(x)-f(x-1).
30.1 Set f(i)=i*2.
Do part 30 for i=-250(1)250.
```

This example illustrates, in part 10 and part 20, two equivalent ways of computing (but not displaying) successive differences of elements in the vector f. The program in part 4 tries both forward and backward methods of proceeding through f and compares time of computation for the two.

Since JOSS assigns storage only for array elements that have an assigned value, each time an array is referenced by the program a search for the requested array element must be made. JOSS facilitates forward (increasing index values) searches by remembering the index of the last reference made and starting each new search from that point.

Do part 4 for k=1,2.

Minutes for case 1: .06

Minutes for case 2: .11

5.141

COMPUTING EFFICIENCY—III

```
Delete all.
Recall item 10 (tbase).
Done.
Type all.

4.1 Line.
4.2 Reset timer.
4.3 Do part k.
4.4 Type k,timer in form 1.

Form 1:
Minutes for case _:  __.__

1.1 Type sum(i=1(1)n: a+i ).
2.1 Type sum(i=1(1)n: 1.23456789+i).
n=1000
a=1.23456789
Do part 4 for k=1,2.

sum(i=1(1)n: a+i ) = 501734.817
Minutes for case 1:     .05

sum(i=1(1)n: 1.23456789+i) = 501734.817
Minutes for case 2:     .06
```

Numeric literals given in JOSS computations require time for conversion to internal form in proportion to their length in characters. In case 2 at the left, the number 1.23456789 is converted for each value of i in the summation, while in case 1 only one conversion was required when a was input.

DEGREE–RADIAN CONVERSION FORMULAS

```
A stored program displays the formulae and sample usage.  *
Do part 1.
r:degrees to radians
h:time to radians
d:radians to degrees.
t:radians to time.
s:sum of two angles

      D(r,c):  ip(r•c)+.01•ip[60•fp(r•c)]+.006•fp(60•r•c)
      R(d,c):  [ip(d)+ip(100•fp[d])/60+fp(100•d)/36]/c
        d(x):  D(x,45/arg(1,1))
        h(x):  R(x,3/arg(1,1))
        r(x):  R(x,45/arg(1,1))
      s(x,y):  d(r(x)+r(y))
        t(x):  D(x,3/arg(1,1))


Some examples:

  r(45.1753) =          .790600213
d(.790600213) =       45.1753
  h(13.4739) =         3.61130437
t(3.61130437) =       13.4739
t(r(23.2951)) =        1.33594
d(h(7.0205)) =       105.3115
s(11.4937,26.3351) = 38.2328
s(21.4703,-39.5702) = -18.0959
```

5.16

BOOLEAN FUNCTIONS

This program displays the values of the 16 boolean functions of two variables.

```
Type all parts, all forms.

1 Page if $≠1.
1.1 Type all formulas,form 1,_.
1.2 Do part 2 for x=true,false.
1.3 Page.

2 Do part 3 for y=true,false.

3 Type x,y,a,b,c,d ,e,f,g,h,i,j,k,l,m,n,o,p in form 2.


Form 1:
  x     y     a b c d e f g h i j k l m n o p

Form 2:
_____ _____ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Do part 1.
```

5.17

```
a:  |x|
b:  |y|
c:  |x=y|
d:  |x≠y|
e:  |true|
f:  |false|
g:  |not x|
h:  |not y|
i:  |x or y|
j:  |x and y|
k:  |not x or y|
l:  |not y or x|
m:  |not(x or y)|
n:  |not x and y|
o:  |not y and x|
p:  |not(x and y)|
```

The 16 boolean functions.

5.171

x	y	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
true	true	1	1	1	0	1	0	0	0	1	1	1	1	0	0	0	0
true	false	1	0	0	1	1	0	0	1	1	0	0	1	0	0	1	1
false	true	0	1	0	1	1	0	1	0	1	0	1	0	0	1	0	1
false	false	0	0	1	0	1	0	1	1	0	0	1	1	1	0	0	1

Values of the 16 functions.

PROGRAM CHAINING

```
Recall item 11 (mastr).
Done.
Type all.
```

Master program in chain.

```
1.05 Type "Fetching the vector f from the files.".
1.1 Recall item 12 (fdata).
1.2 Do part 2 for i=1(1)100.
1.3 Delete f.
1.35 Type "Fetching the vector g from the files.".
1.4 Recall item 13 (gdata).
1.5 Do part 3 for i=1(1)100.
1.52 Discard item 13 (gdata).
1.54 File g as item 13 (gdata).
1.6 Type "New values for g have been computed and filed.".
```

Data chain.

```
2.1 Set s(i)=sum(k=i-10(1)i+10:f(i) )/21.
```

Compute moving averages.

```
3.1 Set g(i)=g(i)+s(i).

10.05 Do part 1.
10.1 Delete part 1,p art 2, part 3.
10.15 Type "Recalling a second processing program from the files.".
10.2 Recall item 14 (2prog).
10.3 Do part 1.
```

```
Delete all.
Recall item 14 (2prog).
Done.
Type all.

1.1 Type  sum(i=1(1)100:g(i)).
```

Auxiliary program.

```
Delete all.
Recall item 11 (mastr).
Done.
Do part 10.
Fetching the vector f from the files.
Fetching the vector g from the files.
New values for g have been computed and filed.
Recalling a second processing program from the files.
sum(i=1(1)100:g(i)) = 30300
```

Sample execution.

5.181

ROOT FINDING

```
Delete all.
Recall item 3 (ex3).
Done.
Type all.

1 Type r(x),p(r(x)) in form 1.

Form 1:
x = __._____     p(x) =  _._____

        i(x):  x-p(x)/q(x)
        p(x):  x*3-10•x*2-6•x+10
        q(x):  [p(x+d)-p(x)]/d
Do part 1 for x = -10,1,10.                    r(i(x))]
x = -1.24670     p(x) =    .00000
x =   .76528     p(x) =    .00000
x = 10.48142     p(x) =    .00000
```

Find the roots of the polynomial p(x) by Newton's method expressed as i(x), where q(x) is the approximate derivative of p(x); r(x) recursively improves the root until a value sufficiently close to zero is obtained.

GAUSSIAN INTEGRATION

```
Recall item 4 (ex4).
Done.
Type all.


        I(f):  h/2•sum(i=1(1)n: sum[j=1(1)m: f(x(i,j))])
           h:  (b-a)/n
      x(i,j):  a+h/2•[t(j)+2•i-1]

           m =         2
           n =        30
a=0
b=1
t(1) = 1/sqrt(3)
t(2) = -t(1)
Type exp(1)-1, I(exp).
    exp(1)-1 =         1.71828183
      I(exp) =         1.71828184
```

PROBABILITY INTEGRAL

This example illustrates a complete JOSS program, including commands to control output formatting, and the results of its execution.

```
Type all.

1.13 Type form 2.
1.15 Do part  2 for b=.1(.1)4.

2.05 Set a=-b.
2.1 Line if fp($/5)=1/5.
2.6 Type b,exp(b),log(exp(b)),c*I(f) in form 1.

Form 1:
_._  .......  _._   _._

Form 2:
  X     EXP(X)     LOG      PROB

        I(f):  h/2*sum(i=1(1)30:sum[j=1(1)2:f(y(i,j))])
        f(x):  exp(-x*2/2)
           h:  (b-a)/30
     y(i,j):  a+h/2*[t(j)+2*i-1]

          c =          .398942281

       t(1) =          .577350268
       t(2) =         -.577350268
```

Do part 1.

X	EXP(X)	LOG	PROB
.10	1.1 00	.100	.0797
.20	1.2 00	.200	.1585
.30	1.3 00	.300	.2358
.40	1.5 00	.400	.3108
.50	1.6 00	.500	.3829
.60	1.8 00	.600	.4515
.70	2.0 00	.700	.5161
.80	2.2 00	.800	.5763
.90	2.5 00	.900	.6319
1.00	2.7 00	1.000	.6827
1.10	3.0 00	1.100	.7287
1.20	3.3 00	1.200	.7699
1.30	3.7 00	1.300	.8064
1.40	4.1 00	1.400	.8385
1.50	4.5 00	1.500	.8664
1.60	5.0 00	1.600	.8904
1.70	5.5 00	1.700	.9109
1.80	6.0 00	1.800	.9281
1.90	6.7 00	1.900	.9426

———, *JOSS: Console Service Routines (The Distributor)*, The RAND Corporation, RM-5044-PR, September 1966.
———, *JOSS: Disc File System*, The RAND Corporation, RM-5257-PR, February 1967.
Marks, S. L., and G. W. Armerding, *The JOSS Primer*, The RAND Corporation, RM-5220-PR, August 1967.
Smith, J. W., *JOSS: Central Processing Routines*, The RAND Corporation, RM-5270-PR, August 1967.

PUBLICATIONS OF HISTORICAL INTEREST

Baker, C. L., *JOSS: Scenario of a Filmed Report*, The RAND Corporation, RM-4162-PR, June 1964.
"The JOSS System: Time-Sharing at RAND," *Datamation*, Vol. 10, No. 11, November 1964, pp. 32–36. (This article is based on RM-4162-PR, above.)
Shaw, J. C., *JOSS: A Designer's View of an Experimental On-Line Computing System*, The RAND Corporation, P-2922, August 1964; also published in *AFIPS Conference Proceedings* (1964 FJCC), Vol. 26, Spartan Books, Inc., Baltimore, Md., 1964, pp. 455–464.
———, *JOSS: Conversations with the Johnniac Open-Shop System*, The RAND Corporation, P-3146, May 1965.
———, *JOSS: Examples of the Use of an Experimental On-Line Computing Service*, The RAND Corporation, P-3131, April 1965.
———, *JOSS: Experience with an Experimental Computing Service for Users at Remote Typewriter Consoles*, The RAND Corporation, P-3149, May 1965.

REFERENCES

1. Baker, C. L., *JOSS: Introduction to a Helpful Assistant*, The RAND Corporation, RM-5058-PR, July 1966.
2. Bryan, G. E., and J. W. Smith, *JOSS Language: Aperçu and Précis, Pocket Précis, Poster Précis*, The RAND Corporation, RM-5377-PR, July 1967.
3. Gimble, E. P., *JOSS: Problem Solving for Engineers*, The RAND Corporation, RM-5322-PR, May 1967.
4. Marks, S. L., and G. W. Armerding, *The JOSS Primer*, The RAND Corporation, RM-5220-PR, September 1967.

INDEX

M

N

O

P

Q

R