MEMORANDUM
RM-5058-PR
JULY 1966

JOSS: INTRODUCTION TO A HELPFUL ASSISTANT

C. L. Baker

PREPARED FOR:
UNITED STATES AIR FORCE PROJECT RAND

The RAND Corporation
SANTA MONICA • CALIFORNIA

MEMORANDUM
RM-5058-PR
JULY 1966

JOSS: INTRODUCTION TO A HELPFUL ASSISTANT

C. L. Baker

This research is sponsored by the United States Air Force under Project RAND—Contract No. AF 49(638)-1700—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 Memorandum should not be interpreted as representing the official opinion or policy of the United States Air Force.

The RAND Corporation
1700 MAIN ST. • SANTA MONICA • CALIFORNIA • 90406

Published by The RAND Corporation

PREFACE

This memorandum was originally delivered as a speech to the Eleventh Annual Data Processing Conference at the University of Alabama Birmingham Center on May 4, 1966.

A brief but explicit description of the capabilities of the JOSS† system is presented through a step-by-step demonstration of the process, with illustrative material taken from actual JOSS output. (The slides prepared for the original presentation have been reproduced and have been inserted in the text of the present memorandum as illustrations.) The unique aspects that distinguish JOSS from other systems permit the user to combine a few highly refined basic features in a variety of ways without restriction. The process is described in layman's terms and will give the uninitiated user, if not the ability to converse fluently with JOSS, the capacity to "overhear" a conversation with almost full comprehension.

This memorandum is a part of The RAND Corporation's continuing program of research in computer sciences under U.S. Air Force Project RAND.

The JOSS system was originally implemented on the JOHNNIAC computer in 1963 by J. C. Shaw, and the present expanded version is implemented on the Digital Equipment Corporation PDP-6 computer.

---

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

SUMMARY

Though JOSS is implemented on a high-speed, general-purpose, time-sharing computer, it is a special-purpose system designed to provide the user with a personal service through remote computation. The only component of the system that the user is aware of is his own console--a mobile unit that is plugged into his office outlet and that supplies computational power. The console itself consists of a standard IBM Selectric typewriter with a slightly modified character set. The conventional characters take up 73 of the standard 88 keyboard positions, leaving 15 positions available for special graphics, which, since the JOSS system is restricted to numeric computations, have been chosen from the usual set of mathematical symbols.

An auxiliary control box, equipped with indicator lights, activates the console. The JOSS console has been designed so that control of the typewriter is proprietary: Either JOSS has control for output purposes, or the user has control for typing in to JOSS. Which of these situations is actually the case is indicated by visual, tactile, and audible signals. The user's input of instructions and data is typed in green, and JOSS responds with output in black.

JOSS commands are limited to one line; take the form of an imperative English sentence, and in fact may be read out loud; begin with a verb; and obey the conventional rules of English for spacing, capitalization, punctuation, and spelling. The ability to append a conditional clause to any JOSS command is an extremely powerful feature of the language. Three brief examples are presented that touch on almost every feature of the JOSS system: the readability of the language, including the identity of the "speaker," its computational ability, its logical ability, and JOSS's response to errors.

In addition to computing directly with numbers, JOSS

can assign values to letters, to help in working with numbers that are repeated many times, or initially have unknown values. Further examples demonstrate that JOSS operates with numbers that are (1) always in decimal, (2) limited to 9 significant digits, (3) are exact on input and output, (4) are expressed in scientific notation where appropriate, and (5) may be denoted by single letters. Any legitimate algebraic expression involving letters, numbers, and functions may replace any number, anywhere in the language, without reservation, and JOSS will interpret the result appropriately. JOSS arithmetic also provides us with the true result, rounded, if necessary, to 9 decimal digits, for the operations of add, subtract, multiply, divide, square root, and selected cases of exponentiation.

Supplementing the basic operations of arithmetic, the JOSS functions fall into three groups: elementary transcendental (log, exp, sin, cos, arg), number dissection (sgn, ip, fp, dp, xp), and iterative (sum, prod, max, min, first).

Several "real" problems are next presented to illustrate how the user can add to JOSS's power to work with him in specific problem-solving situations. We see how JOSS can store values, expressions, functions, and forms, as well as sequences of commands, called steps, for subsequent interpretation.

The ability of JOSS to produce, easily and quickly, report quality output, in a standard format of 8½ by 11 in., contributes a great deal to the power of the system. The value of the JOSS language itself lies not in the user's ability to continually expand and refine the language in many small ways, but in his ability to combine a few highly refined basic features in a variety of ways without restriction. The language is highly readable, and the JOSS user will soon come to actually "think" in the JOSS language--or, at least, to express his problem using JOSS's vocabulary.

ACKNOWLEDGMENTS

I am grateful to the several members of The RAND Corporation staff who assisted in the preparation of the lecture: Ray Clewett for his expertise in taking the slides, Sylvia Comfort for her skill in preparing them, and Edward Lowe and Bernard Dickson for their many valuable suggestions on the organization of the presentation.

CONTENTS

I. GENERAL DESCRIPTION OF JOSS

JOSS is an acronym derived from:

JOHNNIAC--the RAND-built Princeton-type computer, named for the mathematician John von Neumann, on which JOSS was first implemented in 1963.

OPEN SHOP--operation of a computing facility where operating can be performed by any qualified employee of the organization, and not necessarily by the personnel of the computing center itself.

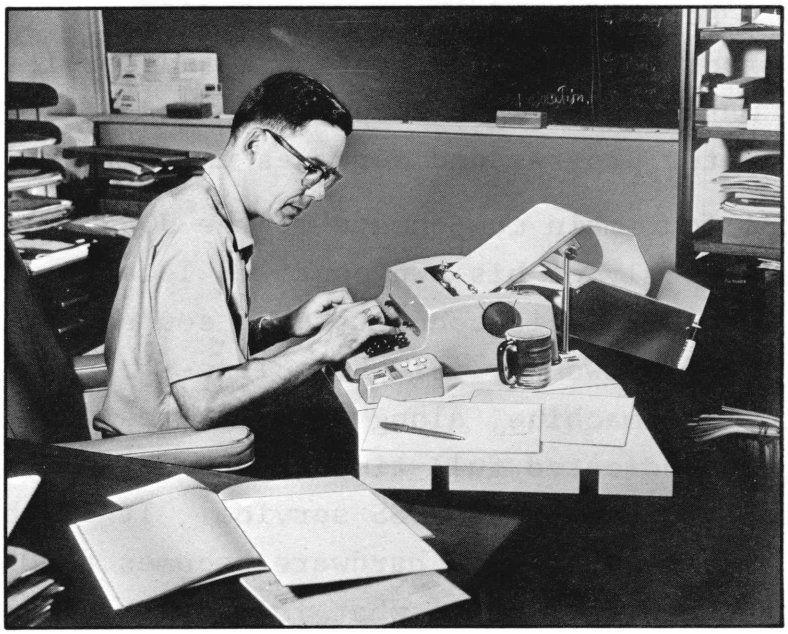
SYSTEM--although "service" would be more descriptive of the goals of JOSS, and, in fact, JOSS is the trademark and service mark of The RAND Corporation for its computer program and services using that program.

Of the four main components of the JOSS system, the user is aware only of his own console, located in his own office. The console is connected, of course, by a communication link--a two-way telephone line--to a central computer. This machine, along with its resident software programs, is dedicated full-time--24 hours a day, seven days a week--to providing JOSS service. It is through the software package that this hardware becomes a JOSS system.

Please keep in mind in what follows that a high-speed, general-purpose, time-sharing computer is used to implement JOSS, but JOSS is not a general-purpose system for time-sharing and using a general-purpose computer. Let me ask you, therefore, to remember that JOSS is a special-purpose system and should be viewed accordingly. It supplies a personal service, and by this we invite comparison with a telephone, a desk calculator, or a slide-rule--always available at a user's desk.

We restrict JOSS to numeric calculations, and do not attempt to provide any of the many symbolic capabilities of the computer. To attract the casual user, and to provide intimate interaction, we have tried to avoid at all costs the placing of many of the small stumbling blocks that computer systems often erect in the path of the would-be user. To the initiate, of course, these pose no problems--in fact, they cannot be seen--but to the novice, these often appear to be barriers beyond which he is hesitant to venture.

This afternoon I am going to introduce you to JOSS by inviting you to look over my shoulder, as you would do as a visitor at RAND, while I demonstrate some of the capabilities of the JOSS system. Since JOSS is <u>not</u> a programming



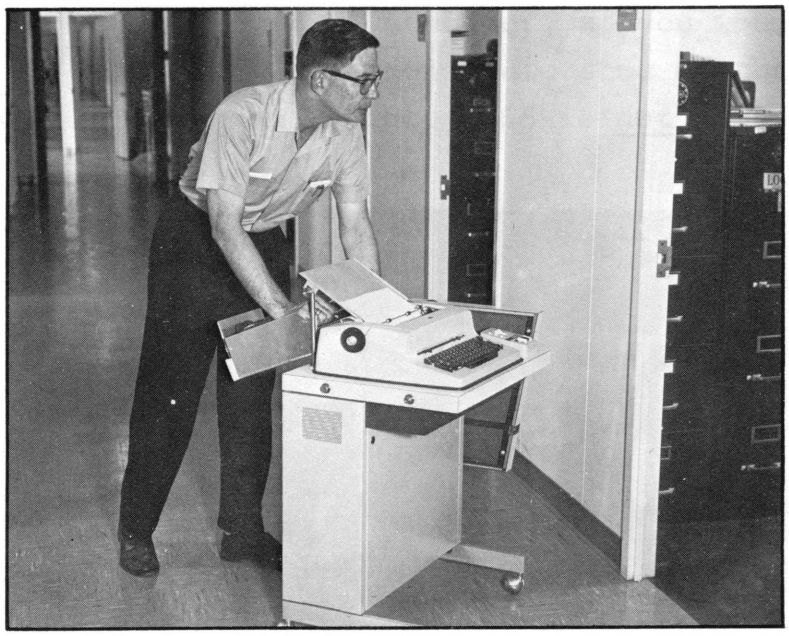
language, and since it is not a language for programmers, I believe that you will leave with the feeling that, while you might not be able to converse with JOSS fluently yourself, you could certainly "overhear" a conversation with almost full comprehension.

Throughout this demonstration, I will try to place

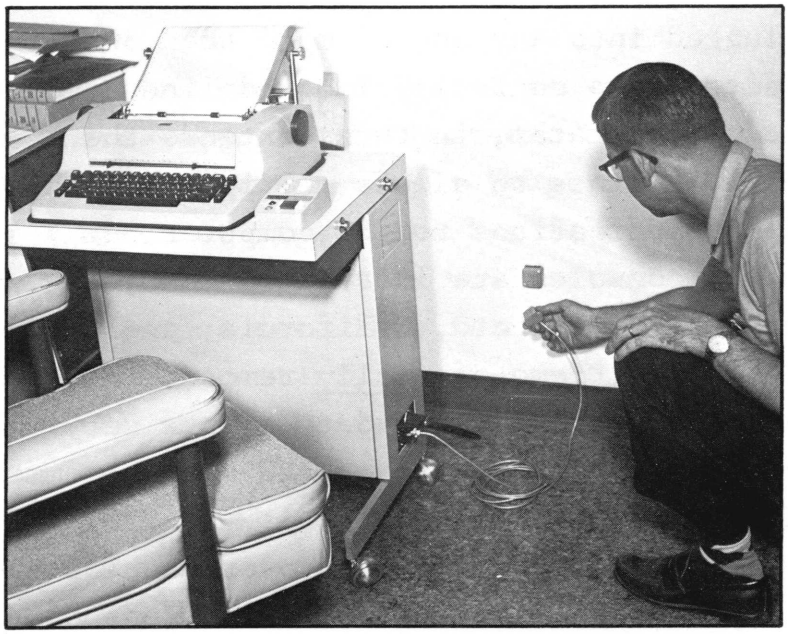
special emphasis on those features that distinguish JOSS from other systems and make it unique--features that have contributed to its enthusiastic adoption by the RAND staff.

II. CONSOLES

The first of the special features of the JOSS system is the JOSS console itself. This unit was developed for this use alone, and, as you will see, is quite different from, say, the familiar Teletype consoles, which were originally developed for an entirely different purpose. The console is mobile, and may easily be moved, as required, from one office to another.

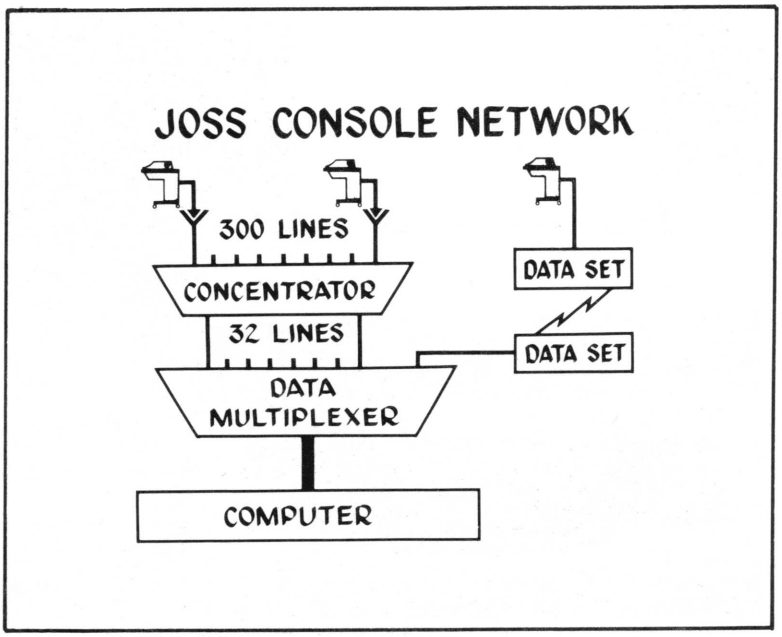


If JOSS were indeed a truly personal service, each user would, of course, have his own console, always available but normally unused, just as with his telephone. Microelectronic circuits notwithstanding, however, the cost of a console is still much higher than that of a telephone, and so too few consoles are available to provide one in each office. What we have done instead is to put a "JOSS Plug" into the office of each RAND staff member. This outlet may be thought of as supplying JOSS computational



power, just as the conventional AC outlet is a source of electrical energy.

As soon as it is plugged in, our console is ready for use; but first I'd like to take a moment to describe the JOSS console network. Each of our 30 JOSS consoles

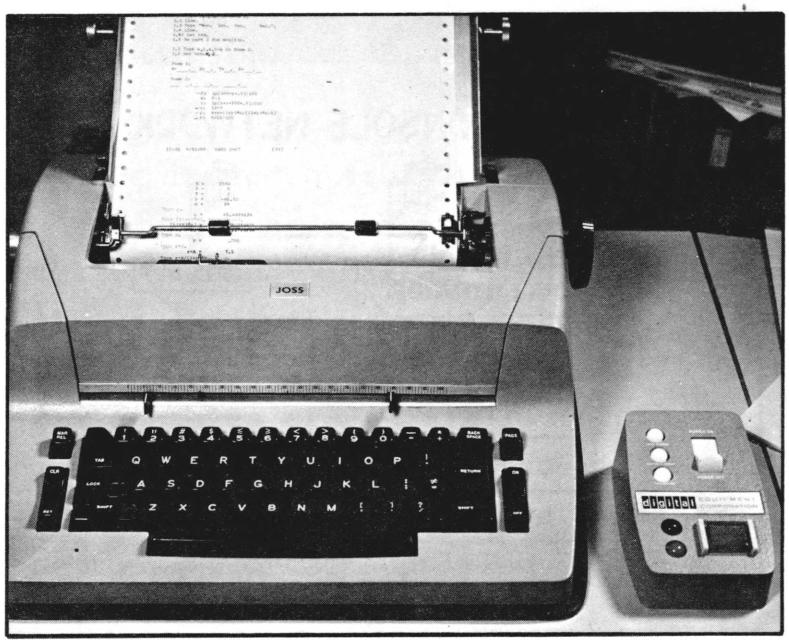


may be plugged into any one of more than 300 lines, which are connected to a centrally located line concentrator. This line concentrator, in turn, extends the connection over one of 32 lines to a data multiplexer, which intermixes all communications to the computer.

Not all consoles are located in RAND, however. At McClellan AFB, Sacramento, California, and at the AEC's Nevada Test Site, remote--really remote, in the Nevada desert--consoles are connected to a data set, which is connected to a matching set at RAND over a common carrier link (AT&T or Western Union lines).

At all times, communication over each of these lines is full duplex--that is, in both directions simultaneously--so that the consoles and the computer can be in immediate contact with each other.

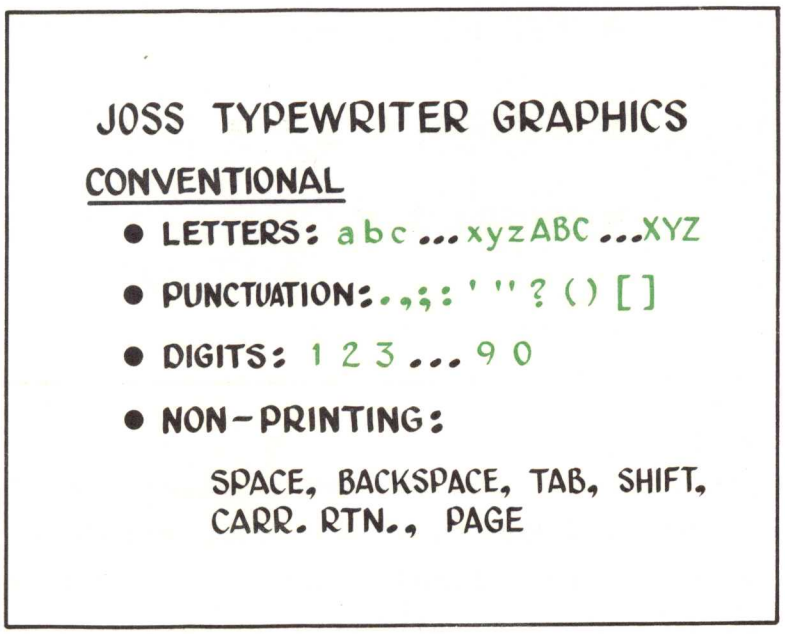
We're now ready to use our JOSS console. But just as we look over the dashboard of a new or unfamiliar auto before we start the motor and drive away, let's spend a few moments for a "cockpit layout check."



KEYBOARD

The typewriter keyboard itself is undoubtedly familiar to everyone here--even to the hunt-and-peck artist. We can try out the typewriter simply by pushing the ON switch and typing a bit. At first, the "golf-ball" typing element of the IBM Selectric typewriter will attract the attention of anyone who has not seen this fascinating mechanism at work. It is more important, however, to become accustomed to the "feel" of the keyboard and to locate the conventional and unique characters available.

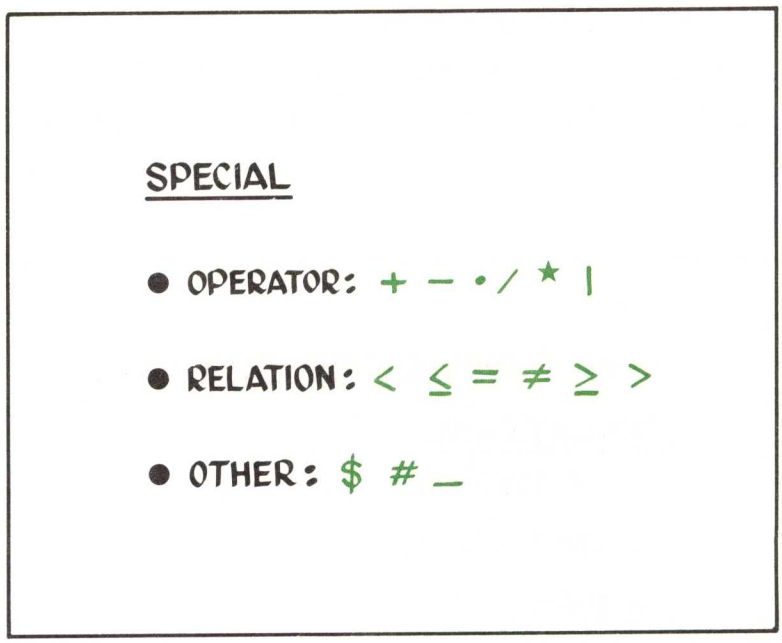
We'll return to the auxiliary control box in a moment, but first let's take a closer look at the typewriter graphical character set.



The letters (upper and lower case) preserve their conventional positions, as do most of the usual punctuation marks. The period and comma in the upper case position have been replaced by brackets, and we should point out that lower case "one" and "ell" are different, as are "oh" and "zero." The nonprinting functions--SPACE, BACKSPACE, TAB, SHIFT,

and CARRIER RETURN--are all to be found in their usual positions, and do the usual things. PAGE is a RAND-added feature that combines forms-feed with carrier return.

These conventional characters take up 73 of the standard 88 keyboard positions, leaving 15 positions available for special graphics which, since the JOSS system is designed for numeric computations, have been chosen from the usual set of mathematical symbols.

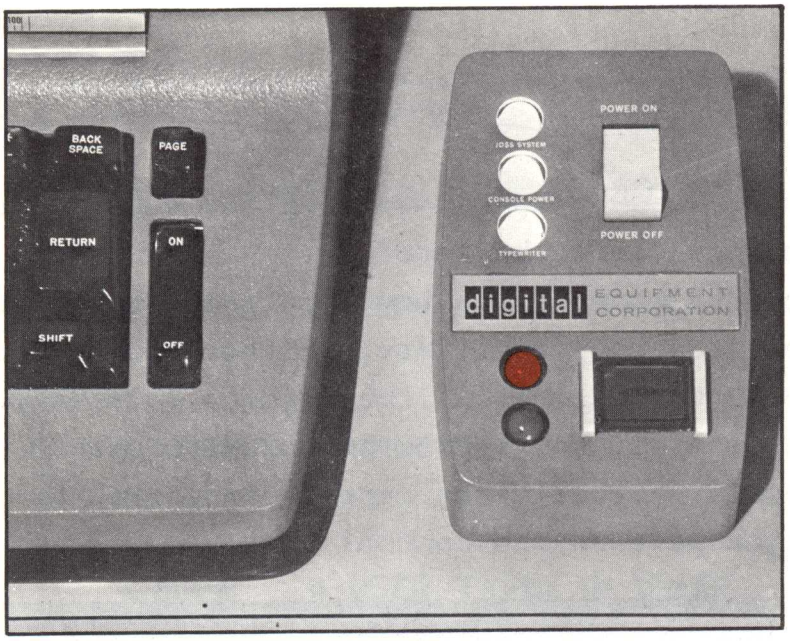


The standard centered dot has been selected to indicate multiplication, and the slash has been retained to indicate division. Mathematical notation commonly uses a superscript to indicate raising a number to a power; to linearize such notation requires an explicit sign, for which we chose a 5-pointed, upward-pointing, elevated asterisk (drawn on the following slides as a star). The relations "less than," etc., round out the mathematical signs. The remaining characters are all used for special purposes. The dollar sign is always appropriate; the "#"

sign is used to strike out any typing errors; and the use of the underscore will be shown later.

CONTROL BOX

Now that we've familiarized ourselves with the typewriter keyboard, we can take a closer look at the auxiliary control box on the console.



Of primary concern is the POWER ON switch, which we use to request JOSS service. When power has been applied, three white status lights let us know in turn that

1. The console electronics package is working.
2. The JOSS system is working.
3. The typewriter is working.

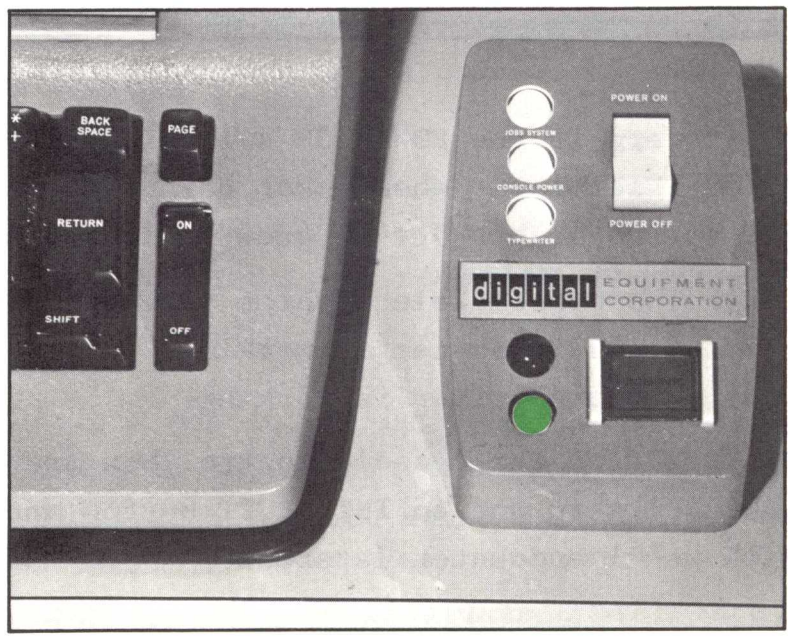
In the lower half of the console control box are a RED light, a GREEN light, and an INTERRUPT button and light, all of which we'll encounter later on.

We've seen where the controls are, so let's try to use JOSS. As we turn on the console with the POWER ON switch, the typewriter comes to life and types out:

JOSS II at your service.
Initials please:

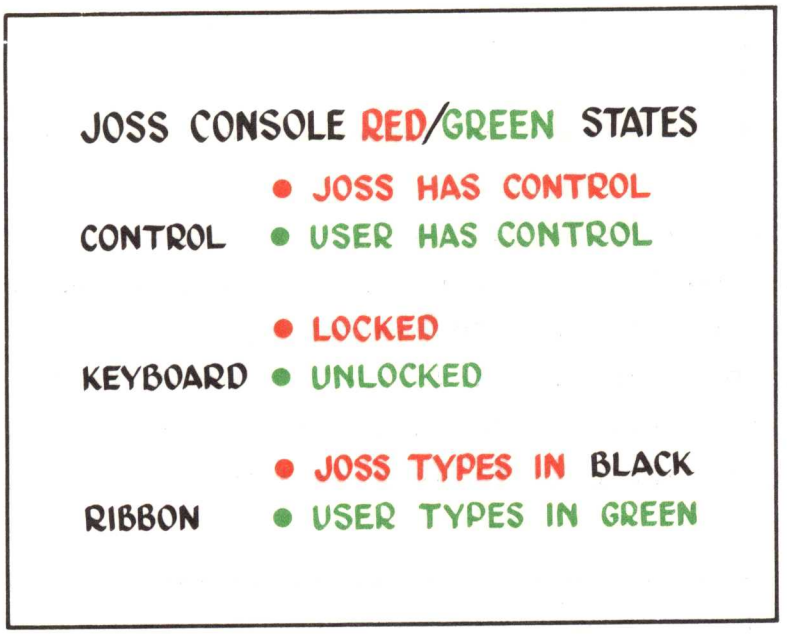
It looks as if JOSS is now waiting for some action on our part, but how can we be sure?

The JOSS console has been designed so that control of the typewriter is proprietary: Either JOSS has control for output purposes, as in the sign-on salutation, or the user has control, for typing in to JOSS. Which of these situations is actually the case is indicated by the RED/GREEN light pair on the control box.



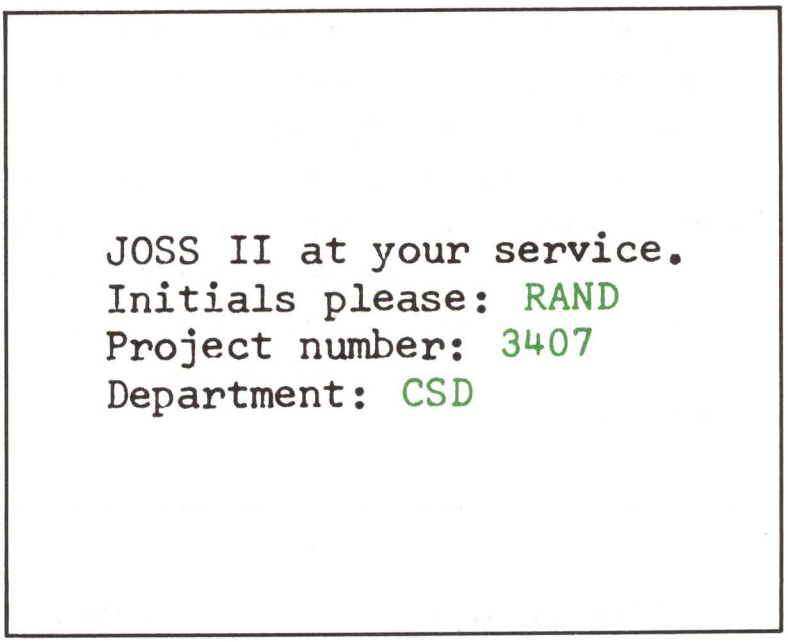
We therefore speak of the console being either in the RED state (computer control) or, as shown above, in the GREEN state (user has control and may type).

When JOSS turns control over to the user, the following signals are given: (1) light changes to GREEN, (2) keyboard unlocks, (3) ribbon changes to GREEN, and (4) soft BEEP tone is sounded. These visual, tactile, and audible signals leave no doubt in the user's mind as to who is in control of the station.

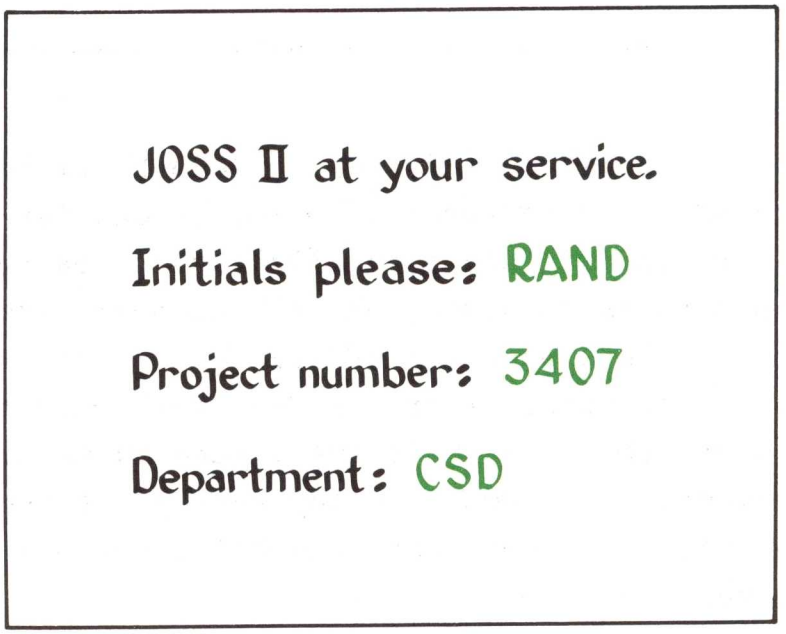


The user turns control back to JOSS by either CARRIER RETURN or PAGE with consequent motion of carrier and platen. At that time, the signals are: (1) light goes to RED, (2) ribbon changes to black, and (3) keyboard locks.

The two colors used for typing, black and green, not only leave a permanent record of "who said what to whom," but form a valuable aid while the system is being used. For the moment, we'll not make any typing mistakes, or forget to follow instructions, and we'll respond properly to JOSS's requests.

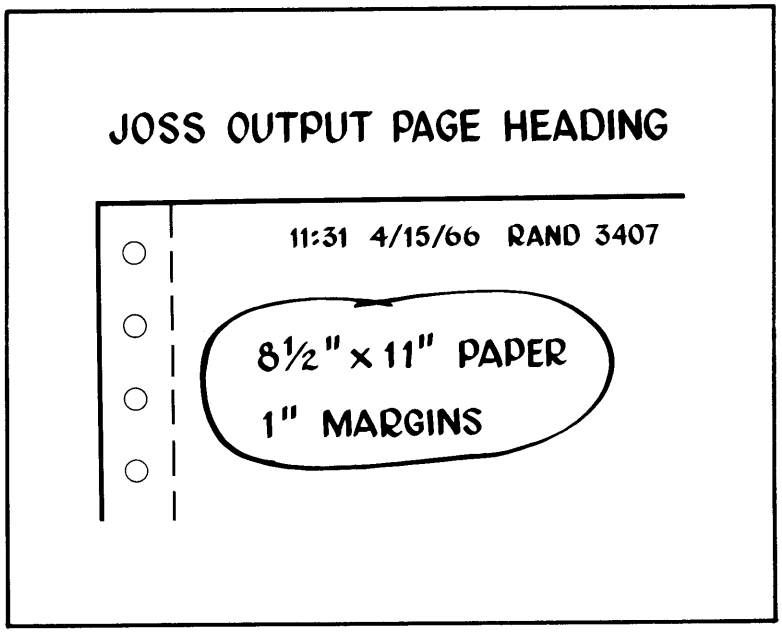


What is shown above is reproduced from actual typewriter output. Now it turns out that typewritten material is fine for reading at a desk or console, or for putting into reports, but not for slides in a lecture hall. So, I've had the following slide, and the rest of them as well, produced by our graphic arts department.



III. COMPUTATION

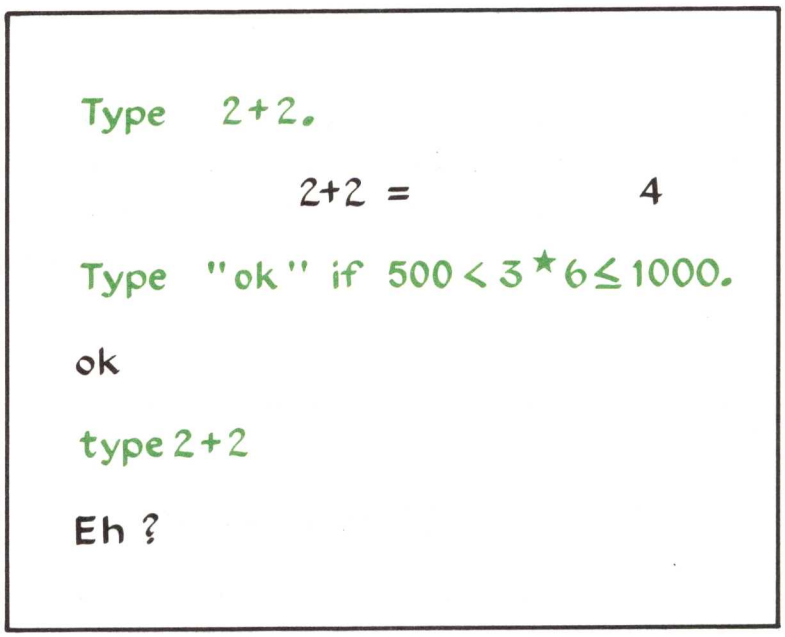
As the sign-on procedure is completed with the final carriage return, JOSS advances the form in the typewriter to the top of a new sheet of paper, where the time, date, and user identification are typed. Also on each sheet is a centered page number.



The output paper is pin fed, but when the tear strips are removed, each sheet, with a heading line as shown above, is reduced to the standard size of 8½ by 11 in. In addition, JOSS feeds enough lines to leave a 1-in. margin at the top of each page, and will also give us a 1-in. margin at the left and 1 in. at the bottom of each page.

This unique feature of JOSS is especially appropriate for RAND, since as a nonmanufacturing research organization, almost all of our tangible product consists of 8½ by 11 in. sheets of typewritten paper.

As a starter, let's ask JOSS for the answer to that simplest of all problems--what is 2+2? We do this by asking JOSS to Type the value of 2+2.



JOSS responds, in black, with "2+2 = 4", as we would expect.

In the second example, which is read "Type "ok" if 500<3*6≤1000.", we observe that a JOSS command may be modified by a conditional clause. In this case 3*6 = 729, which meets the stated condition, so the remainder of the command is obeyed, and JOSS types "ok"; this could have been any series of characters. Had the stated condition not been obtained, JOSS would simply have returned control without typed response. The third example shows JOSS's response to meaningless inputs (compare with correct form of first example). The error response "Eh?" covers a great variety of situations; the user can quickly scan the offending line to determine the cause of error. Thus JOSS avoids misleading the user by misconstruing his intentions. Other error situations give rise to more particularized error messages, as we will shortly see, but the brevity of the "Eh?" response to an obvious error will be appreciated by the user.

Notice that a command to JOSS is

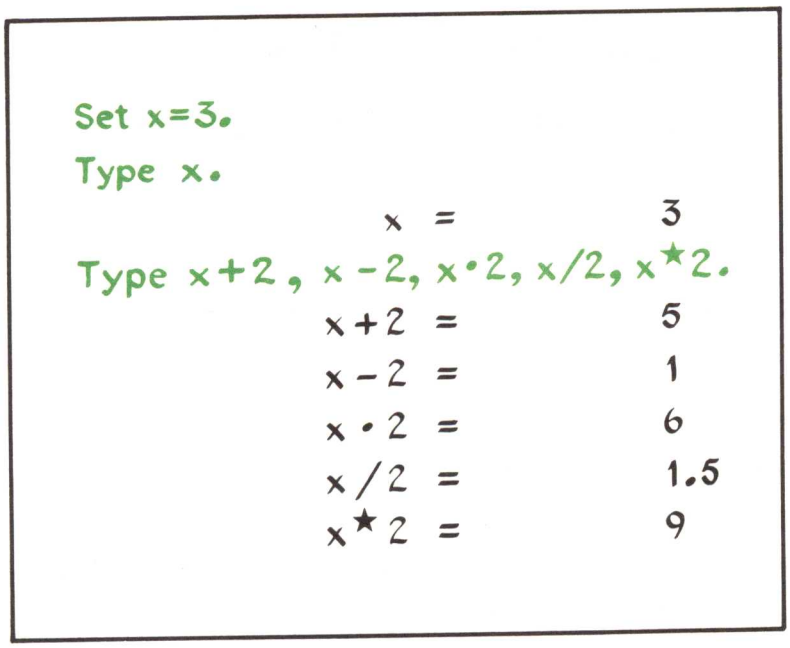
1. Limited to one line.
2. Takes a form of an imperative English sentence, and in fact may be read out loud.

3. Begins with a verb.
4. Obeys the conventional rules of English for spacing, capitalization, punctuation, and, of course, spelling.

The ability to append a conditional clause to any JOSS command is an extremely powerful feature of the language. We shall not use it further this afternoon, since an introductory demonstration is hardly the occasion for untangling complicated logical knots.

In these three brief examples, on one slide, we have touched on almost every feature of the JOSS system: the readability of the language, including the identity of the "speaker," its computational ability, its logical ability, and JOSS's response to errors. And, incidentally, we have tested a very large portion--perhaps 90 percent--of the hardware and software of the entire JOSS system.

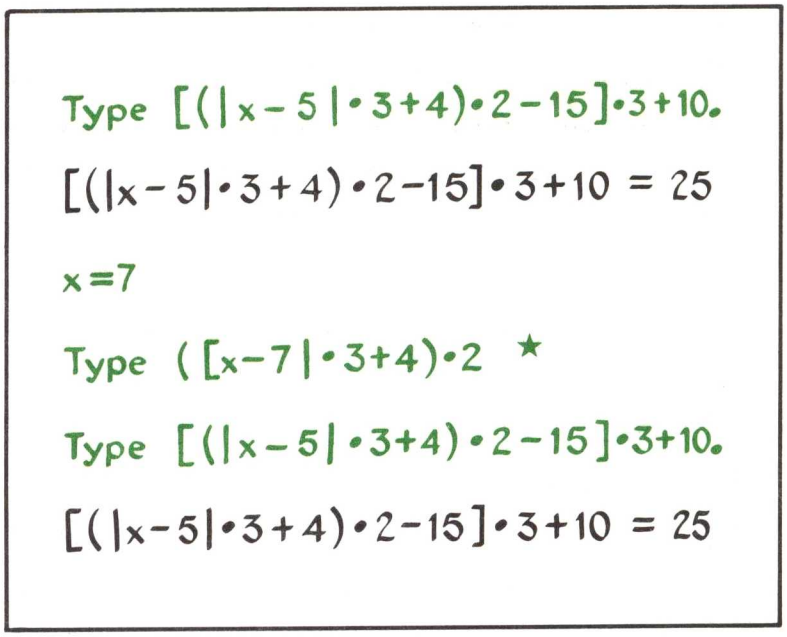
In addition to computing directly with numbers, we can ask JOSS to assign values to letters, to help us work with numbers that are repeated many times, or initially have unknown values. The command "Set x = 3." results in no



output, but JOSS responds by switching back to green. The response to "Type x." verifies that x indeed has been assigned the value 3. Each of the 52 lower and upper case letters may be used this way.

We can compact several calculations onto one line, as above, where we see each of the arithmetic signs used. As always, we can read the command out loud: "Type x+2, x-2, x·2, x/2, x to the power 2 (or simply x squared)." Each result appears, as we see, on a separate output line, with the decimal points aligned.

More elaborate expressions can, of course, be evaluated, although the inclusion of the brackets, parentheses, and absolute value signs required to linearize the expression makes it harder to read.



But let's try: "The absolute value of x minus 5, times 3, plus 4, all times 2, minus 15, all times 3, plus 10 equals 25."

To try another value of x, we merely say "x = 7" (using an abbreviated form for input), and try again. Here, in attempting to type the expression again, I've hopelessly botched

things up. An asterisk instructs JOSS to ignore the line, and we start again. Had the error been less serious, backspacing plus strikeover would have sufficed.

This is an appropriate point to explain that although the nontypist is initially annoyed by JOSS's insistence on capitals, periods, spacing, and correct spelling, he soon learns that their contribution to readability is indeed worthwhile, and that the requirement for the letter-perfect entry of expressions is really the limiting factor of typewriter input--a limitation that is experienced by typist and nontypist alike.

JOSS has a number of features that help us overcome this problem, but for the moment we'll continue with our demonstration of JOSS's computational ability. We use

```
Type sqrt(3),_,sqrt(4).
      sqrt(3) =        1.73205081

      sqrt(4) =        2
Type sqrt(-1).
I have a negative argument for sqrt.
```

"sqrt" as an abbreviation, so we can read "Type the square root of 3, leave a blank line, type the square root of 4." Note the use of parentheses enclosing the 3 and 4. JOSS comes right back with a 9-digit number for the square root of 3, a blank line, and the exact value of the square root

of 4. If we try the impossible operation of a negative square root, JOSS lets us know with an appropriate error message.

JOSS provides a number of elementary functions, such as sine and cosine, shown below, which yield 9 decimal

```
Type sin(.5), cos(.5).
     sin(.5) =          .479425539
     cos(.5) =          .877582562
Type exp(0), exp(1), exp(20).
     exp(0) =           1
     exp(1) =           2.71828183
     exp(20) =          4.85165195*10**8
```

digit results. There are, of course, an infinite number of digits in both of these answers; JOSS has rounded "...386..." to "...39" and "...618..." to "...62". Not shown is the computation $\sin^2+\cos^2$, to which JOSS replies with the single digit "1". The exponential function, "e to the power x", yields an exact answer, 1; the rounded value of e; and, in the case of e to the 20th power, JOSS expresses the answer in scientific notation: "4.85... times 10*8", as is appropriate for very large or for very small numbers.

It is often desirable to be able to work with the various parts of a number expressed in scientific notation. To show this, we give y the value -1.23456·10*2, or -123.456. We can ask JOSS to "Type y, the integer part of y, the

```
y = -1.23456*10*2
Type y,ip(y),fp(y),dp(y),xp(y).
          y =      -123.456
      ip(y) =      -123
      fp(y) =         -.456
      dp(y) =     -1.23456
      xp(y) =      2
```

fractional part of y, the digit part of y, and the exponent part of y." The fp operation, for example, allows us to test to see if a computed number is an exact integer by using the phrase "if fp(x) = 0."

In statistical work, as in many other scientific fields, we often encounter the summation operator, a built-in JOSS function. Shown below is a textbook example that we can ask JOSS to verify: for a value of N =, say, 100. Since the

$$\sum_{i=1}^{N} i^2 = \frac{N(N+1)(2N+1)}{6}$$

```
N = 100
Type sum[i=1(1)N:i*2].
sum[i=1(1)N:i*2] = 338350
Type prod[N,N+1,2*N+1]/6.
prod[N,N+1,2*N+1]/6 = 338350
```

JOSS notation is concise, it's not immediately readable, but an example will clarify: "Sum over i (from 1, in steps of 1, to N) the values of the expression: i squared." The result should be equal to the product of the values of N, N+1, and 2 times N+1, and indeed this is the case. Other similar JOSS functions permit us to find the maximum or minimum of a series of values, or a list of values. These iterative expressions provide a first hint of how we can let JOSS work for us in repetitive calculations.

Before we go on with the JOSS demonstration, I'd like to show you the actual typewriter output, or protocol, of what we have done so far (see facing page), and, at the same time, summarize the features of JOSS that we have observed.

First, JOSS operates with numbers that are

1. Always in decimal.
2. Limited to 9 significant digits.
3. Are exact on input and output.
4. Are expressed in scientific notation where appropriate.
5. May be denoted by single letters.

Also, I must point out that although not explicitly shown in these examples, any legitimate algebraic expression involving letters, numbers, and functions may replace <u>any</u> number, <u>anywhere</u> in the language, without reservation, and JOSS will interpret the result appropriately.

Second, JOSS arithmetic provides us with the <u>true result</u>, rounded, if necessary, to 9 decimal digits, for the operations of add, subtract, multiply, divide, square root, and selected cases of exponentiation.

A few examples will serve to emphasize the care with which JOSS does familiar decimal arithmetic, and, incidentally, will provide you with exercises to try on other systems. In the first example we subtract .05 from 1 followed

11:39 4/15/66 RAND 3407 [1]

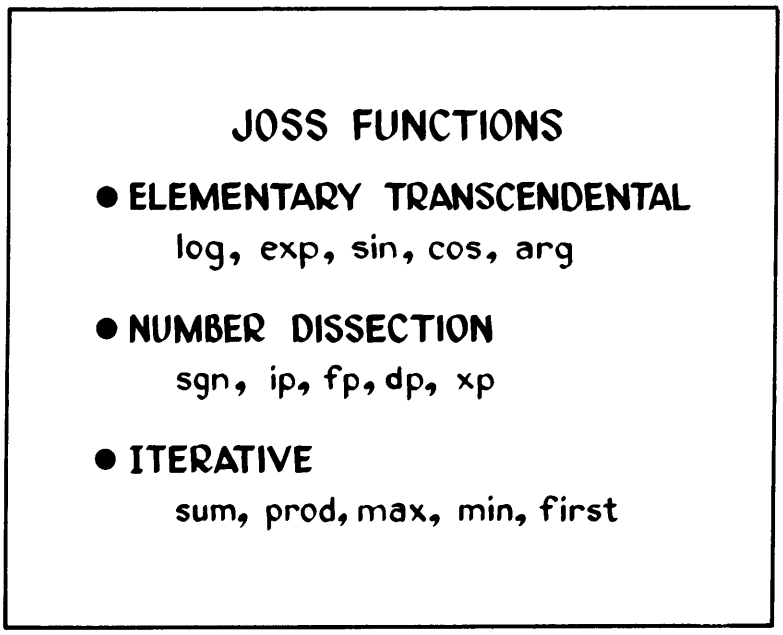
```
Type 2+2.
        2+2 =         4
Type "ok" if 500<3*6≤1000.
ok
type2+2
Eh?
Set x=3.
Type x.
          x =         3
Type x+2,x-2,x•2,x/2,x*2.
        x+2 =         5
        x-2 =         1
        x•2 =         6
        x/2 =         1.5
        x*2 =         9
Type [(|x-5|•3+4)•2-15]•3+10.
[(|x-5|•3+4)•2-15]•3+10 = 25
x=7
Type ([x-7|•3+4)•2 *
Type [(|x-5|•3+4)•2-15]•3+10.
[(|x-5|•3+4)•2-15]•3+10 = 25
Type sqrt(3),_,sqrt(4).
     sqrt(3) =         1.73205081

     sqrt(4) =         2
Type sqrt(-1).
I have a negative argument for sqrt.
Type sin(.5), cos(.5).
     sin(.5) =          .479425539
     cos(.5) =          .877582562
Type exp(0), exp(1), exp(20).
      exp(0) =         1
      exp(1) =         2.71828183
     exp(20) =         4.85165195•10*8
y = -1.23456•10*2
Type y,ip(y),fp(y),dp(y),xp(y).
          y =      -123.456
      ip(y) =      -123
      fp(y) =         -.456
      dp(y) =        -1.23456
      xp(y) =         2
N=100
Type sum[i=1(1)N:i*2].
sum[i=1(1)N:i*2] = 338350
Type prod[N,N+1,2•N+1]/6.
prod[N,N+1,2•N+1]/6 = 338350
```

```
Type 10*8-.0500000000.
10*8-.0500000000 =     1.10*8
Type 10*8-.0500000001.
10*8-.0500000001 =     9.99999999.10*7
Type sqrt(.5).
    sqrt(.5) =          .707106781
Type [sqrt(.5)]*2.
[sqrt(.5)]*2 =          .5
```

by 8 zeroes, or one hundred million, and are not surprised when JOSS returns 10*8 as the rounded answer. However, if we increase the .05 by a single digit in the 10th decimal place, the true answer does not round to 10*8, but is indeed 9.99999999·10*7. Nineteen-digit arithmetic is involved. In the second example, the square root of ½ yields a 9-digit number, and this number, when squared, yields ½--a very comforting situation.

Supplementing the basic operations of arithmetic, the JOSS functions fall into three groups, as shown on the facing page. The accuracy of the logarithmic, exponential, and circular functions is not so easily stated as with arithmetic. Great care is taken to hit "magic" values on the nose, and for most purposes it is correct to say that the resulting values elsewhere are in error by at most a few digits in the last decimal place. The number dissection functions, of course, yield exact answers, while the accuracy of the summation, product, and max and min operators is subject to the arithmetic operations involved as well as the order of, say, summation. The remaining function "first" is essentially a table-look-up operator; since it



properly belongs to "advanced JOSS," it will be skipped over here.

The examples so far have covered all of those operations by which JOSS can directly aid the user. To continue, I've chosen several so-called real problems to illustrate how the user can add to JOSS's power to work with him in specific problem-solving situations.

The first of these (shown on the following page) is taken from a beautiful book by E. H. Lockwood, A Book of Curves,† where we find the formula for s, the arc length along a parabola, as a function of two parameters a and t. Our first thought is to proceed as before, by entering values for a and t and by asking JOSS to type the value of the expression. But JOSS has the ability to store expressions as well as numbers. In this example we see a common subexpression sqrt(1+t*2) appearing twice; we'll denote this by the letter r, for root. We ask JOSS to store our expression by means of the command: "Let r = sqrt(1+t*2)." Similarly, "Let s = a·[t·r+log(t+r)]." We have thus ab-

†Cambridge University Press, Cambridge, 1961, p. 8.

Arc Length Along a Parabola:

$$s=a\left[t\sqrt{1+t^2}+\log\left(t+\sqrt{1+t^2}\right)\right]$$

```
Let r = sqrt(1+t*2).
Let s = a*[t*r+log(t+r)].
Type s.
Error in formula s:  a = ? ? ?
```

breviated the long expression by a single letter; and we ask JOSS to "Type s." An error message reminds us that we have neglected to enter a value for a--and for t as well.

We enter these values and try again, and now JOSS responds with the value of the expression that we have abbreviated by the single letter s.

```
a=3
t=1.5
Type s.
          s =        11.69678
t=2.5
Type s.
          s =        25.1360615
Type formula s.
          s:  a*[t*r+log(t+r)]
```

The same expression may now be evaluated several times for different parameters, and to recall one of our stored expressions, we ask JOSS to "Type formula s." This ability to abbreviate an entire expression by a single letter is appreciated by the typist and nontypist alike, and shows how we have added to JOSS's power for our particular problem.

Even more powerful is the ability to define a complete functional relationship for JOSS, and to illustrate this, I've chosen the classical problem of finding the roots of a polynomial; that is, finding those values of x that make, for example, $x^3-10x^2-6x+10 = 0$. We will have to evaluate the

```
Polynomial Equation:
    x^3-10 x^2-6x+10=0
Let P(x) = x*3-10*x*2-6*x+10.
Type P(-10), P(0), P(10), P(20).
      P(-10) =     -1930
       P(0) =        10
      P(10) =       -50
      P(20) =      3890
```

polynomial many times, so we ask JOSS "Let P of x = ...". JOSS stores the expression as before, but now we can easily indicate the values for which the polynomial is to be evaluated and can, in a single step, "Type P(-10), P(0), P(10), P(20)." I've cleverly chosen the polynomial and its coefficients so that we can see immediately that there are roots between -10 and 0, between 0 and 10, and between 10 and 20. Let's see if we can find the value of x between 0 and 10 for which P(x) = 0.

So far in our demonstration, all of the output from JOSS has been in a standard format--each number is identified and decimal points are aligned. Since we are shortly going to start getting a lot more output from JOSS, we'd like JOSS to use a more economical format, and one that is particularized to our problem. We do this by first entering a form that will describe the output format desired.

```
Form 1:
x = __._____  f(x) = ___._____
x = 5/3
Type x, P(x) in form 1.
x =   1.66667  f(x) = -23.14815
```

Forms are identified, not by letters, but by numbers, as "Form 1:" The colon serves to remind us that the next full line is the form itself; in this form we may type literal information such as "x =" and "f(x) =", and denote fields for decimal numbers by using the underscore combined with the decimal point. After entering a value for x, we ask JOSS to "Type x, P(x) in form 1.", with the results shown above. Notice that JOSS has rounded the value 5/3 to fit into the first field, and that a position is required for the minus sign in the second field. Because of the ease with which we describe output formats to JOSS, we'll use them freely from now on.

We've seen so far how JOSS can store values, expressions, functions, and forms for us; the next step is to get JOSS to store a command, or a sequence of commands, called a part, for subsequent interpretation. We can thus avoid retyping a command many times, and at the same time provide a means for carrying out a number of commands in sequence.

```
1. Type x, P(x) in form 1.

Type step 1.

1. Type x, P(x) in form 1.

Do step 1.

x = 1.66667   f(x) = -23.14815
```

We indicate that JOSS is to store a command by typing it as before, but preceded by a numeric step label, which also serves to identify the step. We can now ask JOSS to "Type step 1.", which JOSS does, or to "Do step 1.", which results in the same action as if we had entered step 1 as a direct command.

The real problem at hand, however, is to find a value of x that makes the polynomial P(x) = 0. We now know that one such value lies somewhere between zero and one; we'd like to repeat step 1 for a number of values of x in this range and see the result. We can ask JOSS, therefore, to "Do step 1 for x = zero, in steps of point two, through one."

```
Do step 1 for x = 0(.2)1.
x =   .00000  f(x) =  10.00000
x =   .20000  f(x) =   8.40800
x =   .40000  f(x) =   6.06400
x =   .60000  f(x) =   3.01600
x =   .80000  f(x) =   -.68800
x =  1.00000  f(x) =  -5.00000
```

JOSS obeys this command, and the result is six lines of formatted output. This narrows the range of x somewhat (to between .6 and .8), and we could continue to home in on x by taking finer and finer intervals over a smaller range by asking, say, "Do step 1 for x = .7(.01).8.", and eventually we would get our answer.

There's a much better way, however, which we can find by consulting any textbook or handbook of numeric methods and where, almost without fail, we will find a description of the Newton-Raphson method of root solving. This method states that if we have a point that is an approximation to the root of an equation, a better approximation is given by subtracting the value of the function at that point divided by the derivative of the function at that point from the approximation. As shown in the next slide, we denote this improved value of x by the function i(x), and copy the rest of the formula directly into JOSS; except that we use Q(x) to denote the derivative P'(x). But what about that derivative? Nearby in our numerical analysis book, we should find the formula for an approximate derivative, and we can also copy it directly into JOSS.

- If x is an approximate root of $P(x) = 0$
- Then $x - \frac{P(x)}{P'(x)}$ is an improved root

Let i(x) = x - P(x)/Q(x).

Let Q(x) = [P(x+d) - P(x)]/d.

That's just about all we need to begin. We'll start with a guess, .7, and ask for the improved value, i(x).

```
x = .7
Type i(x).
Error in formula Q: d = ???
d = .0001
Type i(x), P[i(x)] in form 1.
x =   .76708   f(x) =   -.03519
```

We've forgotten d, of course, so we enter an appropriately small value and try again--but let's use form 1, too. Sure enough, we seem to get a better x--at least f(x) is smaller.

Let me point out what may not be entirely obvious: The literal information (here "x =" and "f(x) =") that we have entered in the form does not in any way determine the numbers that print in the form. This is merely descriptive information that JOSS types along with our numbers.

Thus, x being a guess, and i(x) being a better guess, if we improve the improved value with i[i(x)], that ought to be better still--at least it seems to be.

```
Type i[i(x)].

      i[i(x)] =          .765280012

Let R = i(i[i(i[i(x)])]).

Type R, P(R) in form 1.

x =    .76528  f(x)  =  .00000
```

But what we need is to improve our guess a number of times, so we will use the letter R as an abbreviation for "the improved value of, the improved value of, the improved value of ... etc.," five times--which should be enough for our purposes. And, when we try it, we find that R indeed yields a root of the polynomial from the repeated application of the basic definition of Newton's method, as copied from our handbook.

Our first look at the polynomial roughly located three roots, but we have found only one so far. So, we'll ask JOSS to store a step to type out R, the root derived from

starting with a guess, x, near the root, and the value of the polynomial for that root.

```
2. Type R, P(R) in form 1.
Do step 2 for x = -1, 1, 10.
x = -1.24670  f(x) =    .00000
x =   .76528  f(x) =    .00000
x = 10.48142  f(x) =    .00000
Delete all.
```

Then we "Do step 2" for three initial guesses for x: -1, 1, and 10. JOSS responds with the values of the three roots, which appear to be correct to at least 5 places.

Our part in the solution of the problem has been merely to identify the formulas to be used; we have delegated to JOSS not only the arithmetic computations required, but also all of the sequencing of these computations, and even the formatting of the output.

The entire sequencing problem cannot always be turned over to JOSS so neatly, however; and besides, the problem we have just solved has a somewhat artificial air about it. In several years of demonstrating JOSS to many different groups and individuals with every possible background, I've found only one problem with wide appeal--a problem that everyone has, that almost no one can solve, and that is universally understood: the amortization of a loan in equal monthly payments. Most of us have the problem as debtors, to be sure, but even the creditors are interested.

The first step is easy: We borrow (or think about borrowing) money to buy a new car, perhaps. When we look up the formula for monthly payments to check the quoted interest rate, we are confronted with a formidable-looking equation involving small numbers raised to large powers, which is almost impossible to evaluate by hand--and so we take the dealer's word.

$$p = \frac{A \cdot r \cdot (1+r)^N}{(1+r)^N - 1}$$

```
Let p = A*r*(1+r)*N/[(1+r)*N-1].
Let r = [R/12]/100.
A = 1500
R = 7
N = 24
```

But with JOSS, we start off by copying the formula directly, or almost; we do have to add brackets to put the expression on one line. In this formula the interest rate, r, is per period (in our case, per month), but we would prefer to work directly with a percent per year--6 or 7 or 10 or whatever; so we must also define r to be the yearly percentage; r is divided by 12 to get the monthly percentage, and by 100 to get the true rate.

The remaining values we enter directly: the amount "A = 1500", the rate "R = 7", and the term "N = 24". We "Type p." and learn that our payments will be $67.1588831 monthly. But we should really get rid of those fractional

```
Type p.
                p =       67.1588831
Let P = ip[100*p+ .5]/100.
Form 1:
A=_______ R=___ N=___ P=_____
Type A,R,N,P in form 1.
A= 1500.00 R= 7.0 N= 24 P= 67.16
```

pennies, and we can define a new P, which will round to the nearest penny. We multiply p by 100 to express the value in pennies, add ½ cent to round up, use the "integer part" function to keep a whole number of pennies only, and then divide by 100 to get back to dollars and cents.

We'll probably want a formatted output if we plan to try, say, several terms, rates, or amounts; so, as before, we define a form for our output and ask JOSS to type the amount A, the rate R, the term N, and the payments P. We could easily, as before, store this step, and ask JOSS to "Do" it for a number of values of A, R, or T.

A more interesting problem, however, is that of actually constructing an amortization table to keep track of how our balance is declining, and how much interest we are paying in a year. Our payments are first applied to the interest due, and what's left over goes to reduce the balance. The interest due is merely the rate times the existing balance. As before, we more or less copy these definitions into JOSS (rounding the interest amount to the penny), after which we enter forms 2 and 3 (see following page). Form 2 has no fields for numbers--just column headings for month, interest, reduction

```
P = i + a
i = r • B
Let a = P-i.
Let i = ip[r•B•100 + .5]/100.
Form 2:
Mo.   Int.   Pr.        Bal.
Form 3:
---  --.--  ---.--  ------.--
```

of principal, and balance remaining. In the same way, form 3 has fields for these values that are lined up directly under the respective headings.

Our remaining task is to store a step that will type these values in form 3: month, m; interest, i; reduction of principal, a; and balance, B. When we try to "Do step 1

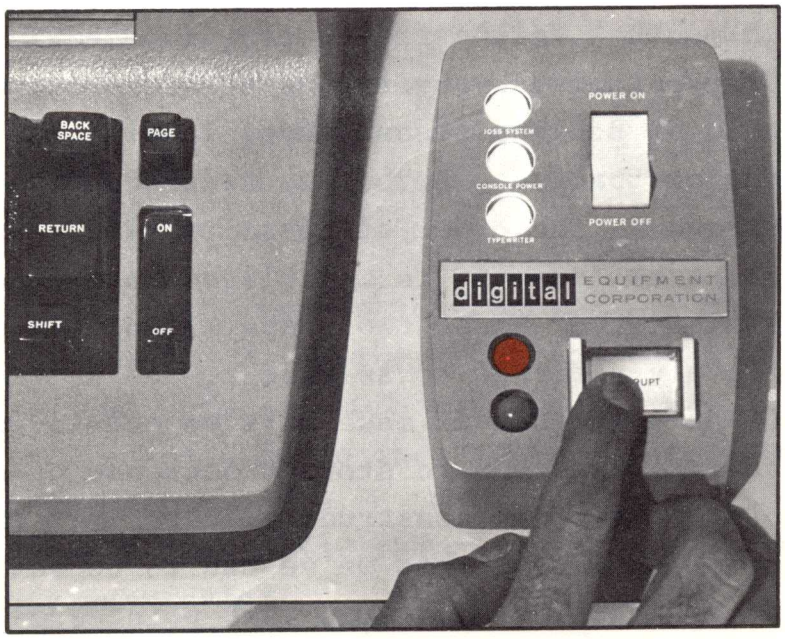
```
1 Type m,i,a,B in form 3.
Do step 1 for m = 1(1)N.
Error at step 1 (in formula i)  B = ???
B = A
Go.
```

starting m (the number of the month) with one, then increased in steps of one to N (the period of the loan)," we get an error message: JOSS wants to know the value of B, the balance outstanding. Initially, the value of B is the amount (A) of the loan; therefore, B = A. "Go." tells JOSS to resume, and our table begins to emerge.

```
 1   8.75   58.41   1500.00
 2   8.75   58.41   1500.00
 3   8.75   58.41   1500.00
 4   8.75   58.41   1500.00
```

This is clearly wrong; the interest and reduction of principal amounts seem to be OK for the first month, but the principal hasn't been reduced, and all the lines are the same. The month number does go up, but this certainly isn't a very good way to pay off a loan.

For the first time, we're really aware of the fact that the console is in the RED state: the keyboard is locked, the light is red, and unless we regain control, JOSS will supply us with 24 lines of garbage--or 360, if we've tried a 30-year home loan. We must somehow ask JOSS to turn control of the console back to us, and we do this by pressing the INTERRUPT button. JOSS acknowledges this request by lighting the button, as shown below, and shortly returns the



console to the GREEN state, in an orderly manner, with an appropriate message.

```
 5    8.75  58.41   1500.00
 6    8.75  58.41   1500.00
I'm at step 1.
Delete step 1.
1.1 Type m, i, a, B - a in form 3.
1.2 Set B = B - a.
Type part 1.
```

It's apparent that our single stored step isn't up to the job, and that we'll need a two-step procedure to construct our table--one step to type out the table entry itself, and another to reduce the balance by the correct amount each month. In the first step, labeled 1.1, we correct our first mistake by typing B-a, the reduced balance, instead of B. The second step, labeled 1.2, actually replaces the old balance, B, by the reduced balance of B-a. We mustn't reverse the order of these two steps, however, since the interest due, i, is calculated on the basis of the old balance due. These two steps, labeled 1.1 and 1.2, are collectively referred to as part 1, and we can ask JOSS to type out part 1, just to make sure we got it in OK.

Now that we know how to store a sequence of steps in a part, we might as well construct another part that will combine all of the steps that we've had to do so far. We'll use part 2, by labeling each step "Two point something."

```
1.1 Type m, i, a, B - a in form 3.
1.2 Set B = B-a.
2.1 Type A,R,N,P in form 1.
2.2 Line.
2.3 Type form 2.
2.4 Line.
2.5 Do part 1 for m=1(1)N.
2.45 Set B = A.
```

Let's read out loud:

"2.1 Type A,R,N,P in form 1."
"2.2 Line." Leave a blank line on the output page.
"2.3 Type form 2." Form 2 is our column heading form--as yet unused.
"2.4 Line." Another blank line.
"2.5 Do part 1 for m = 1(1)N." The "Do" command is the same as before, except that now we use "part" in place of "step" to refer to all of the steps comprising part 2, taken in sequence.

But before that last step is interpreted, we must set the initial balance, B, to the amount of the loan, A, as JOSS reminded us before. JOSS will insert step 2.45 between steps 2.4 and 2.5, so that it will indeed be interpreted at the proper place.

Now, if all goes well, the single command "Do part 2." will produce our table:

```
Do part 2.
A= 1500.00  R= 7.0  N= 24  P= 67.16

Mo.    Int.     Pr.      Bal.

 1     8.75     58.41    1441.59
 2     8.41     58.75    1382.84
 3     8.07     59.09    1323.75
 4     7.72     59.44    1264.31
 5     7.38     59.78    1204.53
```

Our table looks good, and had I released the "Do part 2." line with the PAGE key, or better yet, added the command "Page." as a step at the beginning of part 2, our table would have been neatly typed on a fresh sheet with the proper margins, as shown in the slide of the actual output on the facing page.

However, at the end of the table, we find a balance line of $-.03. Exactly zero would be much better, of course, and by adding one or two more steps we could without much difficulty arrange to indicate that the final payment should be adjusted by a few pennies. Further, we would undoubtedly want to total the interest for twelve-month periods, for tax purposes, and add similar refinements. None of these will require us to learn new features of JOSS--in fact, in our few examples this afternoon, we have used <u>all</u> of the features of the JOSS language, with only a few minor exceptions.

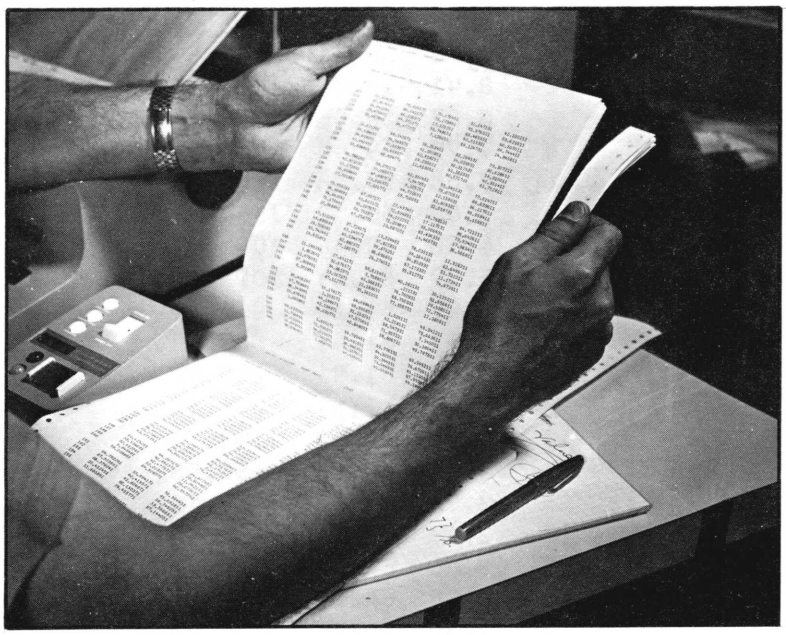
12:23 4/15/66 RAND 3407 [11]

A= 1500.00 R= 7.0 N= 24 P= 67.16

Mo.	Int.	Pr.	Bal.
1	8.75	58.41	1441.59
2	8.41	58.75	1382.84
3	8.07	59.09	1323.75
4	7.72	59.44	1264.31
5	7.38	59.78	1204.53
6	7.03	60.13	1144.40
7	6.68	60.48	1083.92
8	6.32	60.84	1023.08
9	5.97	61.19	961.89
10	5.61	61.55	900.34
11	5.25	61.91	838.43
12	4.89	62.27	776.16
13	4.53	62.63	713.53
14	4.16	63.00	650.53
15	3.79	63.37	587.16
16	3.43	63.73	523.43
17	3.05	64.11	459.32
18	2.68	64.48	394.84
19	2.30	64.86	329.98
20	1.92	65.24	264.74
21	1.54	65.62	199.12
22	1.16	66.00	133.12
23	.78	66.38	66.74
24	.39	66.77	-.03

IV. CONCLUDING REMARKS

The ability to produce, easily and quickly, report quality output, in a standard format, contributes a great deal to the power of the JOSS system.



The power of the JOSS language itself lies not in the user's ability to continually expand and refine the language in many small ways, but in his ability to combine a few highly refined basic features in all manner of ways, without restriction. As you have seen, the language is highly readable, and the JOSS user soon comes to actually "think" in the JOSS language--or, at least, to express his problem using JOSS's vocabulary.

JOSS itself is not a problem-solver, of course, but rather acts as a "helpful assistant" to which the human problem-solver can delegate most of his computational chores. The JOSS language is not a programmer's language, and, in fact, most of the matters with which a programmer usually busies himself--input and output; converting,

branching, looping, testing; compiling and assembling; symbol manipulation and list processing; etc.--are either delegated to JOSS or are entirely out of the scope of the JOSS system. To the professional computer programmer, therefore, JOSS undoubtedly appears naive and unsophisticated. Many of the problems that JOSS users have solved, however--including simultaneous nonlinear integral equations--would tax the abilities of most such programmers. To the large body of its many users at RAND, the JOSS system represents a manyfold increase in power over what is available to them by any other means.