Title

stata.com

input — Enter data from keyboard

Description

`input` allows you to type data directly into the dataset in memory.

For most users, `edit` is a better way to add observations to the dataset because it automatically adjusts the storage type of variables, if required, to accommodate new values.

Quick start

Create numeric `v1`, `v2`, and `v3`, and input data directly into Stata

```
input v1 v2 v3
```

As above, but create `v1` and `v2` as type `int`, `v3` as type `byte`

```
input int (v1 v2) byte v3
```

Add data on string `v4` of length 10

```
input str10 v4
```

Input data for all existing variables

```
input
```

As above, but add observations by typing strings associated with value labels of existing variables instead of numeric data

```
input, label
```

Syntax

`input` [*varlist*] [, `automatic` `label`]

Options

`automatic` causes Stata to create value labels from the nonnumeric data it encounters. It also automatically widens the display format to fit the longest label. Specifying `automatic` implies `label`, even if you do not explicitly type the `label` option.

`label` allows you to type the labels (strings) instead of the numeric values for variables associated with value labels. New value labels are not automatically created unless `automatic` is specified.

Remarks and examples

stata.com

If no data are in memory, you must specify a *varlist* when you type `input`. Stata will then prompt you to enter the new observations until you type `end`.

▷ Example 1

We have data on the accident rate per million vehicle miles along a stretch of highway, along with the speed limit on that highway. We wish to type these data directly into Stata:

```
. input
nothing to input
r(104);
```

Typing `input` by itself does not provide enough information about our intentions. Stata needs to know the names of the variables we wish to create.

```
. input acc_rate spdlimit
        acc_rate   spdlimit
  1. 4.58 55
  2. 2.86 60
  3. 1.61 .
  4. end
. _
```

We typed `input acc_rate spdlimit`, and Stata responded by repeating the variable names and prompting us for the first observation. We entered the values for the first two observations, pressing *Return* after each value was entered. For the third observation, we entered the accident rate (1.61), but we entered a period (`.`) for missing because we did not know the corresponding speed limit for the highway. After entering data for the fourth observation, we typed `end` to let Stata know that there were no more observations.

We can now `list` the data to verify that we have entered the data correctly:

```
. list
     +-------------------------+
     | acc_rate   spdlimit     |
     |-------------------------|
  1. |     4.58         55     |
  2. |     2.86         60     |
  3. |     1.61          .     |
     +-------------------------+
```

◁

If you have data in memory and type `input` without a *varlist*, you will be prompted to enter more information on *all* the variables. This continues until you type `end`.

▷ Example 2: Adding observations

We now have another observation that we wish to add to the dataset. Typing `input` by itself tells Stata that we wish to add new observations:

```
. input
        acc_rate   spdlimit
  4. 3.02 60
  5. end
. _
```

Stata reminded us of the names of our variables and prompted us for the fourth observation. We entered the numbers 3.02 and 60 and pressed *Return*. Stata then prompted us for the fifth observation. We could add as many new observations as we wish. Because we needed to add only 1 observation, we typed `end`. Our dataset now has 4 observations.

◁

You may add new variables to the data in memory by typing `input` followed by the names of the new variables. Stata will begin by prompting you for the first observation, then the second, and so on, until you type `end` or enter the last observation.

▷ Example 3: Adding variables

In addition to the accident rate and speed limit, we now obtain data on the number of access points (on-ramps and off-ramps) per mile along each stretch of highway. We wish to enter the new data.

```
. input acc_pts
        acc_pts
  1. 4.6
  2. 4.4
  3. 2.2
  4. 4.7

. _
```

When we typed `input acc_pts`, Stata responded by prompting us for the first observation. There are 4.6 access points per mile for the first highway, so we entered `4.6`. Stata then prompted us for the second observation, and so on. We entered each of the numbers. When we entered the final observation, Stata automatically stopped prompting us—we did not have to type `end`. Stata knows that there are 4 observations in memory, and because we are adding a new variable, it stops automatically.

We can, however, type `end` anytime we wish, and Stata fills the remaining observations on the new variables with *missing*. To illustrate this, we enter one more variable to our data and then `list` the result:

```
. input junk
           junk
  1. 1
  2. 2
  3. end

. list
```

	acc_rate	spdlimit	acc_pts	junk
1.	4.58	55	4.6	1
2.	2.86	60	4.4	2
3.	1.61	.	2.2	.
4.	3.02	60	4.7	.

◁

You can input string variables by using `input`, but you must remember to indicate explicitly that the variables are strings by specifying the type of the variable before the variable's name.

▷ Example 4: Inputting string variables

String variables are indicated by the types `str#` or `strL`. For `str#`, `#` represents the storage length, or maximum length, in bytes of the variable. You can create variables up to `str2045`. You can create `strL` variables of arbitrary length.

For text with only plain ASCII characters, the length in bytes is equivalent to the number of characters displayed. For instance, a `str4` variable has a maximum length of 4, meaning that it can contain the strings `a`, `ab`, `abc`, and `abcd`, but not `abcde`. Unicode characters beyond the plain ASCII range take 2, 3, or 4 bytes each. Thus the same `str4` variable could contain the strings `á`, `áb`, and `ábc`, but not `ábcd` because `á` takes two bytes to store. If you are using `input` with strings containing Unicode characters, you should allow extra room in your `str#` specification. See [U] **12.4.2 Handling Unicode strings**.

Strings shorter than the maximum length can be stored in the variable, but strings longer than the maximum length cannot.

Although a `str80` variable can store strings shorter than 80 characters, you should not make all your string variables `str80` because Stata allocates space for strings on the basis of their *maximum* length. Thus doing so would waste the computer's memory.

Let's assume that we have no data in memory and wish to enter the following data:

```
. input str16 name age str6 sex
                 name        age        sex
  1. "Arthur Doyle" 22 male
  2. "Mary Hope" 37 "female"
  3. Guy Fawkes 48 male
'Fawkes' cannot be read as a number
  3. "Guy Fawkes" 48 male
  4. "Kriste Yeager" 25 female
  5. end

. _
```

We first typed `input str16 name age str6 sex`, meaning that `name` is to be a `str16` variable and `sex` a `str6` variable. Because we did not specify anything about `age`, Stata made it a numeric variable.

Stata then prompted us to enter our data. On the first line, the name is Arthur Doyle, which we typed in double quotes. The double quotes are not really part of the string; they merely delimit the beginning and end of the string. We followed that with Mr. Doyle's age, 22, and his sex, male. We did not bother to type double quotes around the word `male` because it contained no blanks or special characters. For the second observation, we typed the double quotes around `female`; it changed nothing.

In the third observation, we omitted the double quotes around the name, and Stata informed us that Fawkes could not be read as a number and reprompted us for the observation. When we omitted the double quotes, Stata interpreted `Guy` as the name, `Fawkes` as the age, and 48 as the sex. This would have been okay with Stata, except for one problem: `Fawkes` looks nothing like a number, so Stata complained and gave us another chance. This time, we remembered to put the double quotes around the name.

Stata was satisfied, and we continued. We entered the fourth observation and typed `end`. Here is our dataset:

```
. list

     +-----------------------------+
     |          name   age      sex |
     |-----------------------------|
  1. |  Arthur Doyle    22     male |
  2. |     Mary Hope    37   female |
  3. |    Guy Fawkes    48     male |
  4. | Kriste Yeager    25   female |
     +-----------------------------+
```

◁

▷ Example 5: Specifying numeric storage types

Just as we indicated the string variables by placing a storage type in front of the variable name, we can indicate the storage type of our numeric variables as well. Stata has five numeric storage types: `byte`, `int`, `long`, `float`, and `double`. When you do not specify the storage type, Stata assumes that the variable is a `float`. See the definitions of numbers in **[U] 12 Data**.

There are two reasons for explicitly specifying the storage type: to induce more precision or to conserve memory. The default type `float` has plenty of precision for most circumstances because Stata performs all calculations in double precision, no matter how the data are stored. If you were storing nine-digit Social Security numbers, however, you would want to use a different storage type, or the last digit would be rounded. `long` would be the best choice; `double` would work equally well, but it would waste memory.

Sometimes you do not need to store a variable as `float`. If the variable contains only integers between −32,767 and 32,740, it can be stored as an `int` and would take only half the space. If a variable contains only integers between −127 and 100, it can be stored as a `byte`, which would take only half again as much space. For instance, in example 4 we entered data for `age` without explicitly specifying the storage type; hence, it was stored as a `float`. It would have been better to store it as a `byte`. To do that, we would have typed

```
. input str16 name byte age str6 sex
                name        age        sex
  1. "Arthur Doyle" 22 male
  2. "Mary Hope" 37 "female"
  3. "Guy Fawkes" 48 male
  4. "Kriste Yeager" 25 female
  5. end

. _
```

Stata understands several shorthands. For instance, typing

```
. input int(a b) c
```

allows you to input three variables—`a`, `b`, and `c`—and makes both `a` and `b` `int`s and `c` a `float`. Remember, typing

```
. input int a b c
```

would make a an int but both b and c floats. Typing

```
. input a long b double(c d) e
```

would make a a float, b a long, c and d doubles, and e a float.

Stata has a shorthand for variable names with numeric suffixes. Typing v1-v4 is equivalent to typing v1 v2 v3 v4. Thus typing

```
. input int(v1-v4)
```

inputs four variables and stores them as ints.

◁

❑ Technical note

The rest of this section deals with using input with value labels. If you are not familiar with value labels, see [U] **12.6.3 Value labels**.

Value labels map numbers into words and vice versa. There are two aspects to the process. First, we must define the association between numbers and words. We might tell Stata that 0 corresponds to male and 1 corresponds to female by typing label define sexlbl 0 "male" 1 "female". The correspondences are named, and here we have named the 0↔male 1↔female correspondence sexlbl.

Next we must associate this value label with a variable. If we had already entered the data and the variable were called sex, we would do this by typing label values sex sexlbl. We would have entered the data by typing 0s and 1s, but at least now when we list the data, we would see the words rather than the underlying numbers.

We can do better than that. After defining the value label, we can associate the value label with the variable at the time we input the data and tell Stata to use the value label to interpret what we type:

```
. label define sexlbl 0 "male" 1 "female"
. input str16 name byte(age sex:sexlbl), label
                 name        age        sex
  1. "Arthur Doyle" 22 male
  2. "Mary Hope" 37 "female"
  3. "Guy Fawkes" 48 male
  4. "Kriste Yeager" 25 female
  5. end
. _
```

After defining the value label, we typed our input command. We added the label option at the end of the command, and we typed sex:sexlbl for the name of the sex variable. The byte(...) around age and sex:sexlbl was not really necessary; it merely forced both age and sex to be stored as bytes.

Let's first decipher sex:sexlbl. sex is the name of the variable we want to input. The :sexlbl part tells Stata that the new variable is to be associated with the value label named sexlbl. The label option tells Stata to look up any strings we type for labeled variables in their corresponding value label and substitute the number when it stores the data. Thus when we entered the first observation of our data, we typed male for Mr. Doyle's sex, even though the corresponding variable is numeric. Rather than complaining that "male could not be read as a number", Stata accepted what we typed, looked up the number corresponding to male, and stored that number in the data.

That Stata has actually stored a number rather than the words `male` or `female` is almost irrelevant. Whenever we `list` the data or make a table, Stata will use the words `male` and `female` just as if those words were actually stored in the dataset rather than their numeric codings:

```
. list
```

	name	age	sex
1.	Arthur Doyle	22	male
2.	Mary Hope	37	female
3.	Guy Fawkes	48	male
4.	Kriste Yeager	25	female

```
. tabulate sex
```

sex	Freq.	Percent	Cum.
male	2	50.00	50.00
female	2	50.00	100.00
Total	4	100.00	

It is only almost irrelevant because we can use the underlying numbers in statistical analyses. For instance, if we were to ask Stata to calculate the mean of `sex` by typing `summarize sex`, Stata would report 0.5. We would interpret that to mean that one-half of our sample is female.

Value labels are permanently associated with variables, so once we associate a value label with a variable, we never have to do so again. If we wanted to add another observation to these data, we could type

```
. input, label
                name        age        sex
  5. "Mark Esman" 26 male
  6. end

. _
```

❑

❑ Technical note

The `automatic` option automates the definition of the value label. In the previous example, we informed Stata that `male` corresponds to 0 and `female` corresponds to 1 by typing `label define sexlbl 0 "male" 1 "female"`. It was not necessary to explicitly specify the mapping. Specifying the `automatic` option tells Stata to interpret what we type as follows:

First, see if the value is a number. If so, store that number and be done with it. If it is not a number, check the value label associated with the variable in an attempt to interpret it. If an interpretation exists, store the corresponding numeric code. If one does not exist, add a new numeric code corresponding to what was typed. Store that new number and update the value label so that the new correspondence is never forgotten.

We can use these features to reenter our age and sex data. Before reentering the data, we `drop _all` and `label drop _all` to prove that we have nothing up our sleeve:

```
. drop _all
. label drop _all
. input str16 name byte(age sex:sexlbl), automatic
                name        age        sex
  1. "Arthur Doyle" 22 male
  2. "Mary Hope" 37 "female"
  3. "Guy Fawkes" 48 male
  4. "Kriste Yeager" 25 female
  5. end
. _
```

We previously defined the value label `sexlbl` so that `male` corresponded to 0 and `female` corresponded to 1. The label that Stata automatically created is slightly different but is just as good:

```
. label list sexlbl
sexlbl:
             1 male
             2 female
```

❑

Reference

Kohler, U. 2005. Stata tip 16: Using input to generate variables. *Stata Journal* 5: 134.

Also see

[D] **edit** — Browse or edit data with Data Editor

[D] **import** — Overview of importing data into Stata

[D] **save** — Save Stata dataset

[U] 21 Entering and importing data