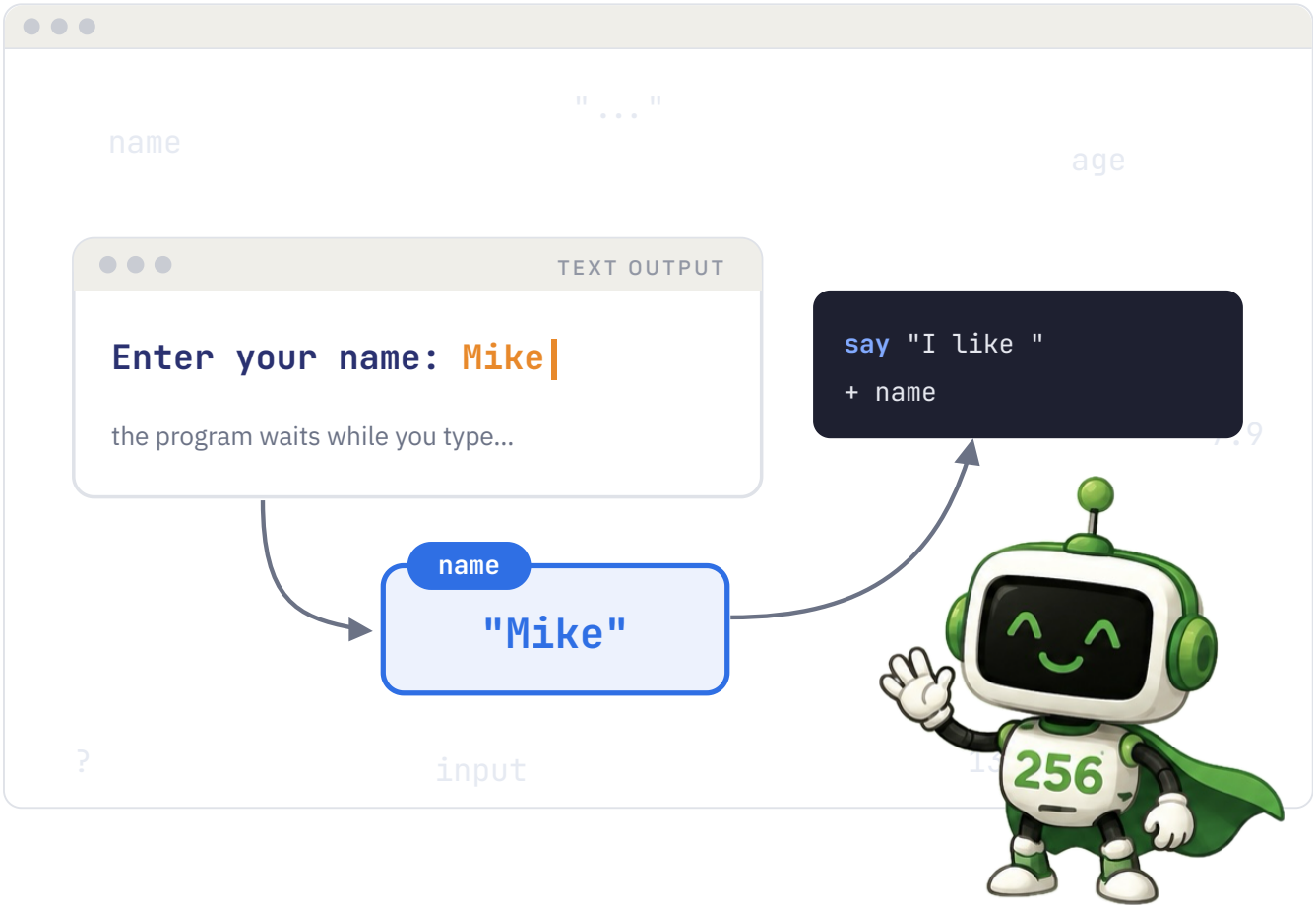
LEARN TO CODE · INPUT

Chapter 6

# Your Program Asks for Your Input

Until now, every value a program used was written into the program itself. In this chapter your programs start asking questions: they wait for you to type a word or a number, store it in a variable and use it — so the same program can give a different answer every time it runs.




© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 1

# Getting text from the user

The `inputstring` statement waits for the user to type something and stores it in a variable.

So far we have told the program everything it needs to know in the code. Program 19 in Chapter 3, for example, always says that hello and yellow rhyme. To make a program that works with *your* words, it has to ask for them.

The next program asks for a name and then tells that person how much it likes them. When you run it, the program stops at the prompt and waits. Type a name in the text output area and press **Enter**.

ilike.kbs — BASIC-256

```
# ilike.kbs - ask for a name and use it
inputstring "Enter your name: ", name
message1 = name + " is my friend."
message2 = "I like " + name + "."

print message1
say message1
print message2
say message2
```

TEXT OUTPUT

```
Enter your name: Mike
Mike is my friend.
I like Mike.
```

says **“Mike is my friend.”** then **“I like Mike.”**

**Program 32** · I Like Fill in the Blank

In the sample outputs, what the user typed is shown in **orange**.




NEW CONCEPT

```
inputstring "prompt", variable
inputstring variable
```

The **inputstring** statement waits for the user to type something into the text output area and press Enter. Whatever was typed is stored, exactly as it was typed, as a **string** in the variable.

If you give a **prompt**, it is printed first and the cursor waits right after it. End the prompt with a space, so that what the user types doesn’t stick to it.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 2

# Getting numbers from the user

When you need a number, use `inputinteger` or `inputfloat`.

The “Math-wiz” program asks for two numbers and shows what you get when you add, subtract, multiply and divide them.

mathwiz.kbs — BASIC-256

```
# mathwiz.kbs - show several mathematical operations
inputfloat "a? ", a
inputfloat "b? ", b

print a + "+" + b + "=" + (a + b)
print a + "-" + b + "=" + (a - b)
print b + "-" + a + "=" + (b - a)
print a + "*" + b + "=" + (a * b)
print a + "/" + b + "=" + (a / b)
print b + "/" + a + "=" + (b / a)
```

TEXT OUTPUT

```
a? 7.9
b? 6
7.9+6.0=13.9
7.9-6.0=1.9
6.0-7.9=-1.9
7.9*6.0=47.4
7.9/6.0=1.31666666667
6.0/7.9=0.75949367089
```

**Program 33** · Math-wiz

The user typed `6`, but the program printed `6.0`: `inputfloat` always stores a floating-point number, even when you type a whole number.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout




NEW CONCEPT

```
inputinteger "prompt", variable
inputinteger variable
inputfloat "prompt", variable
inputfloat variable
```

**inputinteger** stores a whole number (an integer) and **inputfloat** stores a floating-point number. Just like **inputstring**, both can show a prompt first.

**inputinteger** accepts only digits, with an optional `-` or `+` sign in front. **inputfloat** also accepts a decimal point. Spaces before or after the number don't matter, but thousands separators do: `1,234.5` is **not** a number to BASIC-256 — type `1234.5`.

**What if the user doesn't type a number?** If someone types `ten` or `4.5` when **inputinteger** asks for a whole number, the variable gets **0**. What else happens depends on **Preferences ▸ User ▸ Runtime handling of bad type conversions**: with **Ignore**, the standard setting, nothing else happens, so the 0 slips by unnoticed; with **Warn**, you also see *WARNING on line … Unable to convert string to number, zero used.*; with **Error**, the program stops.

Here is another example, using both **inputstring** and **inputinteger**.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

sayname.kbs BASIC-256

```
# sayname.kbs - use a string and an integer from the user
inputstring "What is your name? ", name
inputinteger "How old are you? ", age

greeting = "It is nice to meet you, " + name + "."
print greeting
say greeting

greeting = "In 8 years you will be " + (age + 8) + " years old."
greeting += " Wow, that's old!"
print greeting
say greeting
```

TEXT OUTPUT

```
What is your name? Jo
How old are you? 13
It is nice to meet you, Jo.
In 8 years you will be 21 years old. Wow, that's old!
```

says **“It is nice to meet you, Jo.”** then **“In 8 years you will be 21 years old. Wow, that’s old!”**

## Program 34 · Fancy Say Name

The second greeting is too long for one comfortable line, so it is built in two steps with `+=`, which you met in Chapter 3: the second line adds “Wow, that’s old!” to the end of the string.




**TRY IT**

Take the brackets away, so the line reads `"In 8 years you will be " + age + 8 + " years old."` and run the program again with age 13. Jo is suddenly **138**! Without brackets BASIC-256 works from left to right: it joins the string and 13, which gives a string, and then joins 8 to that. The brackets make it add `age + 8` first.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

SECTION 3

# Input — automatic type conversion

The plain `input` statement looks at what was typed and decides the type by itself.

The last kind of input statement is plain **input**. It takes whatever the user types and converts it: to an integer if it looks like one, otherwise to a floating-point number if it looks like one, and if it is not a number at all, it stays a string. That can be just what you want, but it can also surprise you. The next program shows what happened when it was run three times.

whattype.kbs — BASIC-256

```
# whattype.kbs - let input decide the type
input "Type something: ", thing
print "You typed " + thing
print "It was stored as type " + typeof(thing)
```

TEXT OUTPUT

```
Type something: 007
You typed 7
It was stored as type 1

Type something: 2.50
You typed 2.5
It was stored as type 2

Type something: hello
You typed hello
It was stored as type 3
```

**Program 35** · What Type Is It?

The leading zeros of `007` are gone, and so is the trailing zero of `2.50`: once a number is stored, it is just a number, and those zeros aren't part of it. Remember from Chapter 3 that **typeof** answers 1 for an integer, 2 for a floating-point number and 3 for a string.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout




NEW CONCEPT

```
input "prompt", variable
input variable
```

The **input** statement stores an **integer** if what was typed is a whole number, a **float** if it is any other number, and a **string** otherwise. Spaces around a number, leading zeros and trailing zeros after the decimal point are lost. Even `1e3` counts as a number: it is stored as the float 1000.0.

Like the other input statements, **input** can show a prompt first.

**Which one should I use?** When you know what you need, say so: **inputstring** for words, codes and anything with leading zeros, like a phone number; **inputinteger** for whole numbers; **inputfloat** for any number. Use plain **input** only when the type really doesn't matter.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout




BIG PROGRAM

# Silly Story Generator

This chapter's big program writes a silly story. Answer its questions with words, and the computer tells you a story — out loud.

sillystory.kbs BASIC-256

```
# sillystory.kbs - a silly story generator
print "A Silly Story."

inputstring "Enter a noun: ", noun1
inputstring "Enter a verb: ", verb1
inputstring "Enter a room in your house: ", room1
inputstring "Enter a verb: ", verb2
inputinteger "Enter a whole number 2 or larger: ", howmany
inputstring "Enter a plural noun: ", noun2
inputstring "Enter an adjective: ", adj1
inputstring "Enter a verb in the past tense: ", verb3
inputstring "Enter a noun: ", noun3
inputstring "Enter your name: ", name

sentence = "A silly story, by " + name + "."
print sentence
say sentence

sentence = "One day, not so long ago, I saw a " + noun1
sentence += " " + verb1 + " down the stairs."
print sentence
say sentence

sentence = "It was going to my " + room1 + " to " + verb2
sentence += " " + string(howmany) + " " + noun2 + "."
print sentence
say sentence

sentence = "The " + noun1 + " became " + adj1
sentence += " when I " + verb3 + " a " + noun3 + "."
print sentence
say sentence

sentence = "The End."
print sentence
say sentence
```

**Program 36** · Big Program — Silly Story Generator

Each sentence is built in two steps with `+=`, so that no line gets too long. The number the user typed is turned into a string with `string(howmany)` before it is joined to the sentence; `+` would do that by itself, but `string` makes it clear.

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

TEXT OUTPUT

```
A Silly Story.
Enter a noun: car
Enter a verb: drive
Enter a room in your house: bathroom
Enter a verb: walk
Enter a whole number 2 or larger: 5
Enter a plural noun: cows
Enter an adjective: big
Enter a verb in the past tense: lifted
Enter a noun: hippo
Enter your name: Mary
A silly story, by Mary.
One day, not so long ago, I saw a car drive down the stairs.
It was going to my bathroom to walk 5 cows.
The car became big when I lifted a hippo.
The End.
```

**Sample run** · the program also says each line of the story

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout




PRACTICE

# Exercises

### WORD SEARCH

```
b r m i r g t p a c b t t
r y j n e n o t l o l p t
e j t m s i r h v c m a l
g w y b u r p c w o o o b
e h d c e t o k r l r m r
t f r e r s b p f g c u s
n o p t h t g k v g g j t
i e i n p u t f l o a t r
t p d s o p r e g e t n i
u y b n l n t m h c r k n
p t l p y i p b k d p n g
n r u t u p n i s s s v b
i f h r r m p n r y p o h
```

**Find:** float, input, inputfloat, inputinteger, inputstring, integer, prompt, string, typeof, user




## Problems

Try these — they only use what this chapter and the earlier ones covered.

**1** Write a program that asks for three names and stores them in string variables. Once the user has entered the third name, have the computer print and say the classic playground rhyme using the names:

```
[Name One] and [Name Two]
sitting in a tree,
K-I-S-S-I-N-G.
First comes love,
then comes marriage,
then comes [Name Three]
in a baby carriage!
```

© 2019 James M. Reneau · CC BY-NC-SA 3.0 US · adapted layout

**2** Write a program that asks for an adjective, a noun, an animal and a sound. Once the user has entered the last one, build a **single** string variable (using concatenation and `+=`) holding this verse of “Old MacDonald”. Print it with one statement and say it with one statement.

```
[Adjective] MacDonald had a [Noun], E-I-E-I-O,
and on that [Noun] he had a [Animal], E-I-E-I-O,
with a [Sound] [Sound] here and a [Sound] [Sound] there,
here a [Sound], there a [Sound], everywhere a [Sound] [Sound],
[Adjective] MacDonald had a [Noun], E-I-E-I-O.
```

**3** Ask for the length and the width of a room in metres with `inputfloat`, then print the floor area in square metres. What happens if you type `4,5` instead of `4.5`?

**4** Without running it, predict what Program 35 stores, and as which type, when you type `12`, `12.0`, `012`, `twelve` and `-3.`. Then run it and check.

---

Chapter 6 · Your Program Asks for Your Input. Modernised layout of the BASIC-256 beginner’s guide, updated for BASIC-256 2.3. Programs are numbered 32–36, continuing from Chapter 5, and a new Program 35 shows plain `input` choosing a type. Prompts now end with a space. Corrected for 2.3: `inputfloat` does not accept thousands separators, and a value that isn’t a number becomes 0 silently under the standard *Ignore* setting, with a warning or an error only if you choose them. Long sentences in the Big Program are built with +=, its last sentence gained its full stop, and “Old MacDonald” now has an [Animal]. Every sample output was produced by running the program in BASIC-256 2.3 and typing the answers shown. Original text © 2019 James M. Reneau, licensed under CC BY-NC-SA 3.0 US. This adaptation is shared under the same licence. Mascot: BitBot.