I would like to note that previous answers made many assumptions about the user's knowledge. This answer attempts to answer the question at a more tutorial level.

For every invocation of Python, sys.argv is automatically a list of strings representing the arguments (as separated by spaces) on the command-line. The name comes from the C programming convention in which argv and argc represent the command line arguments.

You'll want to learn more about lists and strings as you're familiarizing yourself with Python, but in the meantime, here are a few things to know.

You can simply create a script that prints the arguments as they're represented. It also prints the number of arguments, using the len function on the list.

```
from __future__ import print_function
import sys
print(sys.argv, len(sys.argv))
```

The script requires Python 2.6 or later. If you call this script print_args.py, you can invoke it with different arguments to see what happens.

```
> python print_args.py
['print_args.py'] 1

> python print_args.py foo and bar
['print_args.py', 'foo', 'and', 'bar'] 4

> python print_args.py "foo and bar"
['print_args.py', 'foo and bar'] 2

> python print_args.py "foo and bar" and baz
['print_args.py', 'foo and bar', 'and', 'baz'] 4
```

As you can see, the command-line arguments include the script name but not the interpreter name. In this sense, Python treats the script *as* the executable. If you need to know the name of the executable (python in this case), you can use sys.executable.

You can see from the examples that it is possible to receive arguments that do contain spaces if the user invoked the script with arguments encapsulated in quotes, so what you get is the list of arguments as supplied by the user.

Now in your Python code, you can use this list of strings as input to your program. Since lists are indexed by zero-based integers, you can get the individual items using the list[0] syntax. For example, to get the script name:

```
script_name = sys.argv[0] # this will always work.
```

Although interesting, you rarely need to know your script name. To get the first argument after the script for a filename, you could do the following:

```
filename = sys.argv[1]
```

This is a very common usage, but note that it will fail with an IndexError if no argument was supplied.

Also, Python lets you reference a slice of a list, so to get *another list* of just the user-supplied arguments (but without the script name), you can do

```
user_args = sys.argv[1:] # get everything after the script name
```

Additionally, Python allows you to assign a sequence of items (including lists) to variable names. So if you expect the user to always supply two arguments, you can assign those arguments (as strings) to two variables:

```
user_args = sys.argv[1:]
fun, games = user_args # len(user_args) had better be 2
```

So, to answer your specific question, sys.argv[1] represents the first command-line argument (as a string) supplied to the script in question. It will not prompt for input, but it will fail with an IndexError if no arguments are supplied on the command-line following the script name.