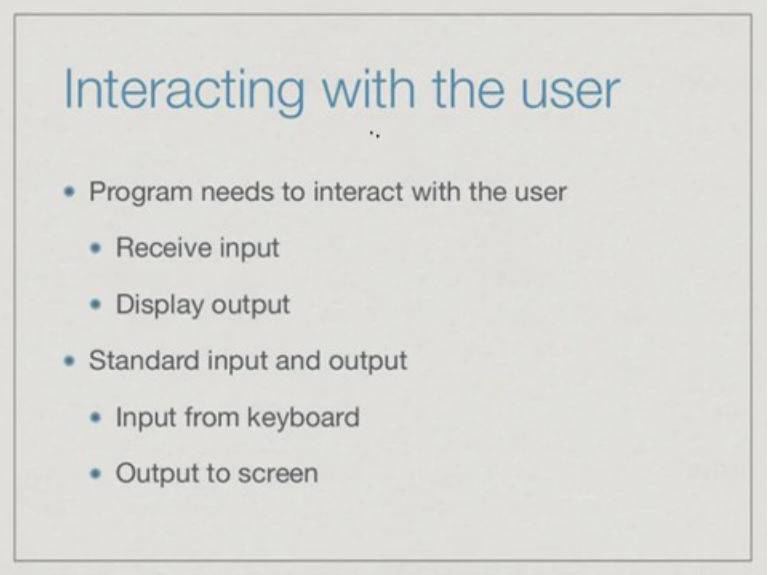
Programming, Data Structures and Algorithms in Python
Prof. Madhavan Mukund
Department of Computer Science and Engineering
Chennai Mathematical Institute,Madras

Week - 05
Lecture - 02
Standard Input and Output

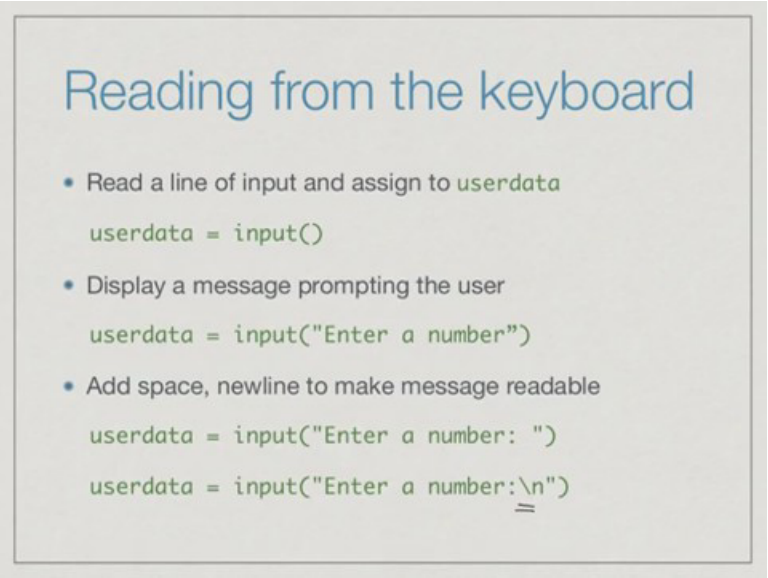
Till now, all the programs that you have been asked to write in your assignments have actually been just functions.

(Refer Slide Time: 00:02)



These are functions, which are called from other pieces of python code and return values to them. Now, when you have a stand-alone python program, it must interact with the user in order to derive inputs and produce outputs. Let us see, how python interacts with its environment. The most basic way of interacting with the environment is to take input from the keyboard and display output to the screen.

(Refer Slide Time: 00:54)



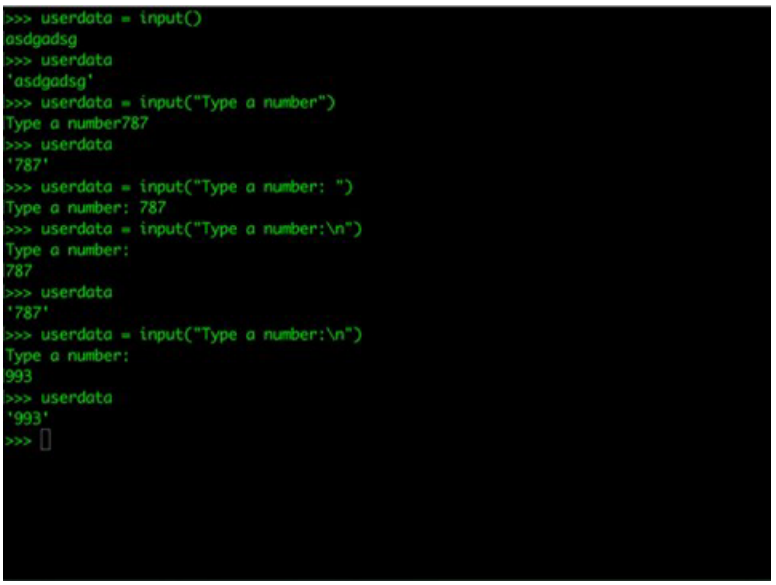
Traditionally, these modes are called standard input and standard output. So, standard input just means, take the input from the keyboard or any standard input device like that and standard output just means display the output directly on the screen. The basic command in python to read from the keyboard is input. If we invoke the function input with no arguments and assign it to a name, then, the name user data will get the value that is typed in by the user at the input command.

Remember that, it reads a line of input. The way that the user signals that the input is over, is by hitting the return button on the keyboard and the entire sequence of characters up to the return, but not including the return, is transmitted as a string to user data. Now, of course, if the program is just waiting for you for input, it can be very confusing. So, you might want to provide a prompt, which is a message to the user, telling the user, what is expected. So, you can provide such a thing by adding a string as an argument to the input. If you put an argument to input like this, then, it is a string which is displayed when the user is supposed to input data.

Now, this string is displayed as it is. So, you can make appropriate adaptations to make it little more user-friendly. We will see an example in a minute, but you might want to leave a space or you might want to insert a new line. Basically, you use the input command to read one line of input from the user and you can display a message to tell the user what is expected of him. So, here is what happens if I just say userdata is equal

to input(), the python program will just wait and now, as a user who does not know what is expected, we do not know whether it is processing something or it is waiting for input.

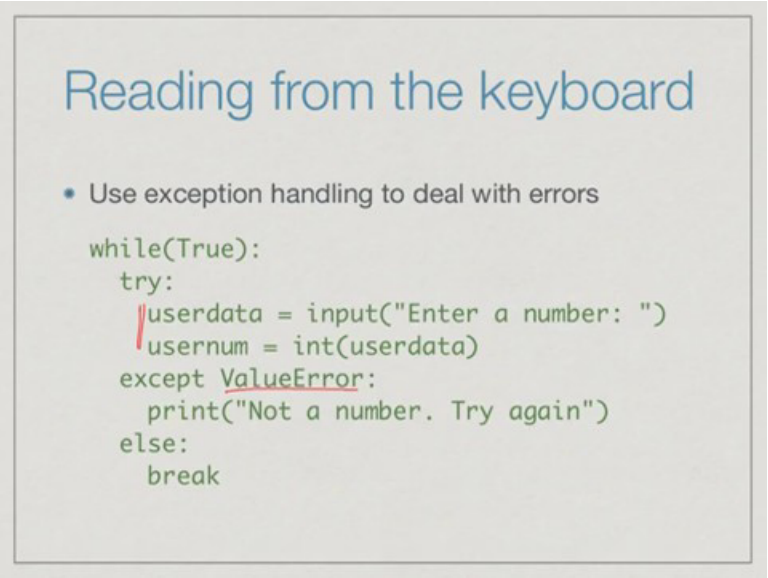
(Refer Slide Time: 02:30)



Now, it so turns out that, if we type something and press enter, it will come back. Now, if I ask for the contents of the name userdata, it will be impact me the string of things that I had typed. So, providing an input prompt without a message can be confusing for the user. So, what we said was, we might want to say something like, provide an input like this. Now, it provides us with a message, but the number that we type for instance, is stuck to the message. It is not very readable. So, userdata is indeed not a number, now, it is a string as we will see in a minute.

But the fact is that, we did not get any space or anything else. It looks a bit ugly. So, what we said is that you can actually, for instance, put a colon and a space so that the message comes like this. Now, this is a slightly nicer prompt and you could also pfirefut a new line if you want, which is signalled by this special character, backslash n. Now, the message comes and then, you type on a new line and in all cases, the outcome is the same; userdata, the name to which you are reading the input, becomes set to the string that is typed in by the user. If I do it again and if I type in something else like 993, for example, then userdata becomes the string “993”. So, you can use input with a message and make the message as readable as you can.

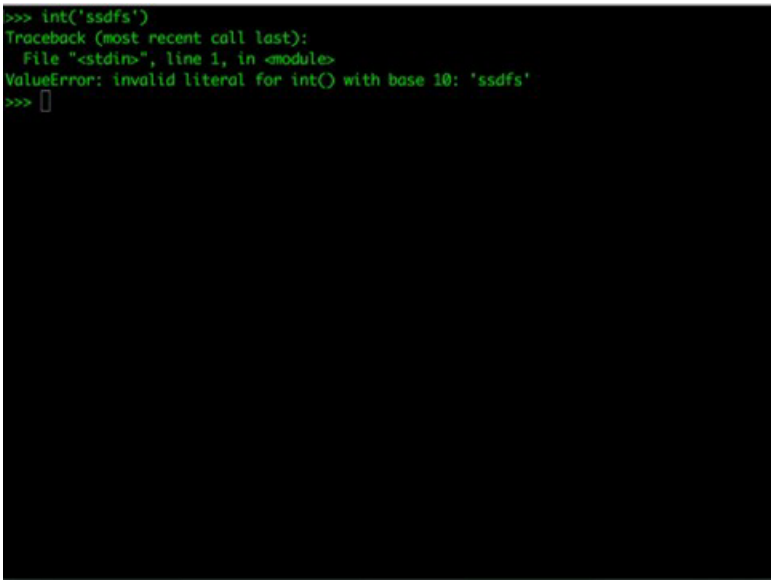
(Refer Slide Time: 04:16)



As we saw, when we were playing with the interpreter, the input that is read by the function input is always a string. Even if you say, enter a number and the user types in a number, this is not actually a number. If you want to use it as a number, you have to use this type conversion. Remember, we have these functions int, str and so on. So, we have to use the int function to type convert whatever the user has typed, to an integer. Now, of course, remember that, if the user types some garbage, then you get an error, right. If the user does not type some, something valid, then you will get an error. So, what we can do is, we can use exception handling to deal with this error.

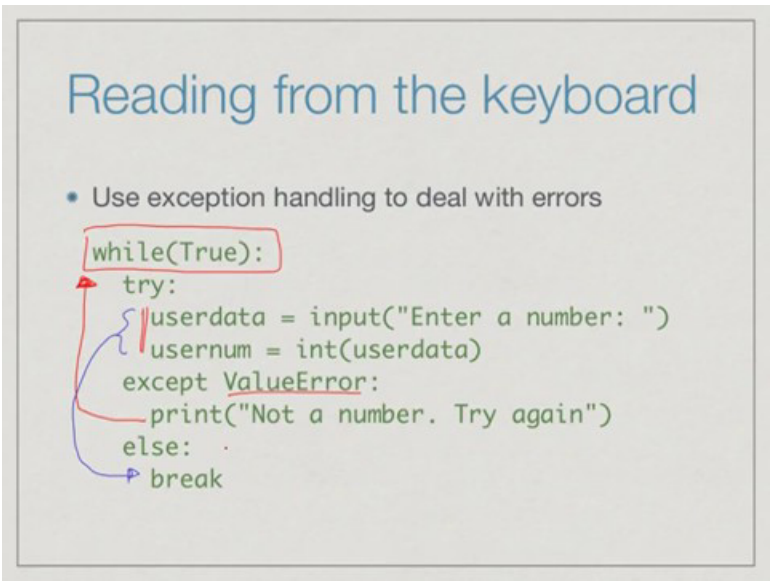
So, what we can say is, try userdata. This is the code that we had before: those 2 lines. So, what we want to say is, we will try these lines, but if the user types something which is not a number, then, we are going to ask him to type it again and it will turn out that, that type of error in python is called a value error.

(Refer Slide Time: 05:20)



So, you can verify this by going to the python interpreter and checking. In the python interpreter, if we try to apply int to some nonsensical things, then we get a value error.

(Refer Slide Time: 05:32)



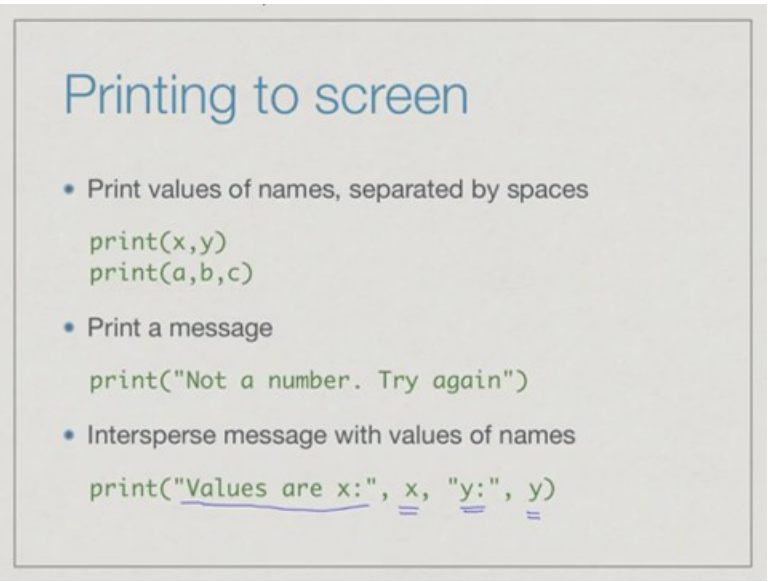
So, what we are trying to say is that, if we get a value error, we want to take appropriate action. So, we have this try block and if we see a value error, what we do is, we print a message. Now, we are going to see print just after this, but we print a message to the user, saying this is not a number, try again and this is now, the whole thing is enclosed inside a while loop and this while loop has a condition True.

In other words, the condition is never going to become false; this while loop is going to keep on asking for a number. So, how do we get out of this? Well, if I come here and I get a value error, it will go back and the while will execute again. but if there is no error, remember, if there is no error here, then it will come to the else. This else is executed if there is no error and what the else does is, it gets us out of this vicious cycle.

In other words, we are in an infinite loop, where we keep on trying to get one more piece of data from the user, until we get something that we like and when we get that, we break out of the loop. This is another kind of idiomatic way to use exceptions, in the context of input and output. As we said in the last lecture, input and output is inherently error prone, because, you are dealing with an uncertain, interacting environment, which can do things, which you cannot anticipate or control. So, you must take appropriate action to make sure that the interaction goes in the direction that you expect.

The other part of interaction is displaying messages, which we call standard output or printing to the screen.

(Refer Slide Time: 06:59)



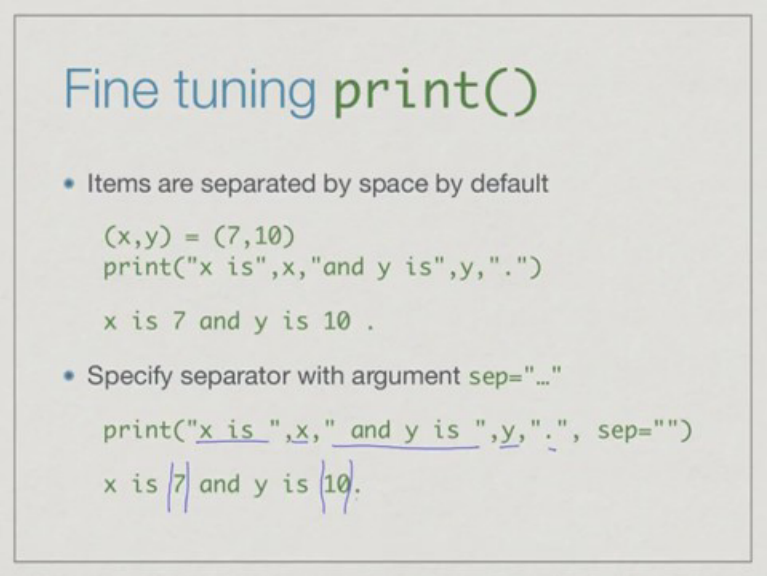
And, this is achieved using the print statement, which we have seen occasionally, informally, without formally having defined it. The basic form of the print statement is to give a sequence of values, separated by commas. So, print x, y will display the value of x, then, a space, then, the value of y. 'print a, b, c' will display 3 values; the values of a,

b and c, separated by spaces. Now, the other thing that we can do is, directly print a string or a message.

Like we saw in the previous example, we can say, print the string “Not a number. Try again”. This will display this string on the screen. Now, you can combine these two things in interesting ways. So, print takes, in general, a sequence of things of arbitrary links, separated by commas. These things could be either messages or names. So, we can say things, supposing, we want to print the value of x and y, but we want to indicate to the output, which is x, which is y.

Instead of saying, just print x comma y, which produces 2 values on the screen, with no indication as to which is x and which is y, we could have this more elaborate print statement, which prints 4 things. The first thing it prints is a message saying, the values are x colon. This will print x colon; it will leave a space; then, it will print the value of, current value of x; then, it will print y colon after a space and then, it will print the current value of y.

(Refer Slide Time: 08:39)



So, we can intersperse messages with values, with names, to produce meaningful output that is more readable. By default, print appends a new line whenever it is executed. In other words, every print statement, we just print the way we have done so far, appears on a new line because the previous print statement implicitly moves the output to a new line. Now, if we want to control this, we can use an optional argument called end. So, we

can provide a string saying, this is what should we put at the end of the print statement; by default, the value here is this new line character. So, we can replace this by something else.

Here is an example. Supposing, we write these 3 statements. The first statement says, ‘print “Continue on the”’; this is just a string; but set end to a space. The second line says, ‘print “same line”’ and then, it says, set end to a full stop and a new line. And then, the third statement says, ‘print “Next line.”’. So, what we are doing is, in the first 2 statements, we are changing the default.

The default would have been to print a new line, but the first statement is not printing a new line. If we print this, what we see is that, the first 2 statements continue on the same line, come on a single line, because, we have disabled the default print new line and we have explicitly put a new line here and this has forced the next one to come on the next line. If we break this up and see what happens here, we see that, in the first statement, we had this end, which says insert a space and this is why we get a space between the word ‘the’ on the first line and the word ‘same’ coming in the second line.

Otherwise, ‘the’ and ‘same’ would have been fused together as a single word, right. So, end equal to space is effectively separating this print from the next print by a space. The next print statement inserts a full stop and a new line. Implicitly, although the word same line ends without a full stop, we produce a full stop and after we produce a full stop, it produces a new line and finally, after this new line, the next line comes in the new line and of course, because here we did not say anything; if we print after that, we implicitly would print on a new line.

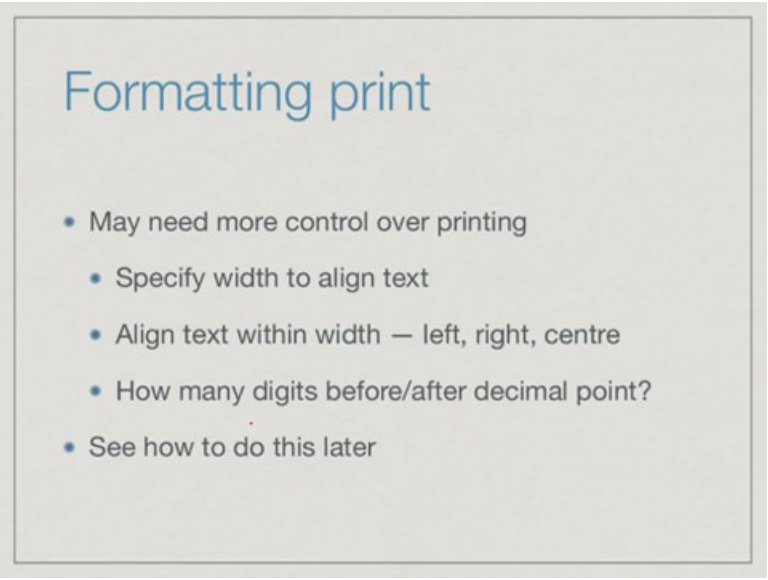
The other thing that we might want to control is how the items are separated on a line. We said that, if we say print x comma y, then x and y will be separated by a space by default, right. If we do this, print x, y, we set x equal to 7, y equal to 10 and we say x is x and y is y and then, we want to end with a full stop. This is what you want; you want to write a string, ‘x is’, then the value of x and ‘y is’, then the value of y and then, a full stop. Now, because everything is separated by a space, what we find is that, we find a space over here; do you see this? This is fine. So, we get a space here, because, that is from this comma; we get a space here, which is from this comma; we get a space here, which is from this comma.

And then, we get an unwanted space between 10 and the full stop. So, how do we get rid of this space, the second space, right? We do not want a full stop to come after the space. So, just like we have the optional argument end, we have an optional argument sep, which specifies what string should be used to separate. So, for example, if we take the earlier thing, we can say, do not separate it with anything.

Now, of course, do not separate it with anything, it changes, because, then, this x is 7 will get fused and this and this will get fused. So, what we do instead is, we put the space explicitly here. Earlier, we had no space here, at the end, just around the quotes. Now, we put spaces where we want them and we say do not put any other spaces. So, what this will say is that, x is space, I give this space; do not put the space, put the value of x; do not put a space, now, I give a space. So and y is, give a space and then, now do not put a space here. These commas do not contribute any space, because I have set separator should be empty and in particular, what this means is that, this last comma, the comma between the y and the full stop, will not generate a space.

And in fact, if you execute this, then, you will get the output, x is 7 and y is 10 and the way it works is that, this is the first block. This is everything up to here. Then, this is the second block, this is this. Then, this is the third block, which is this whole thing, with the spaces given and then, this is the value of y and finally, this is the full stop. This is one way to control the output of a print statement.

(Refer Slide Time: 13:28)

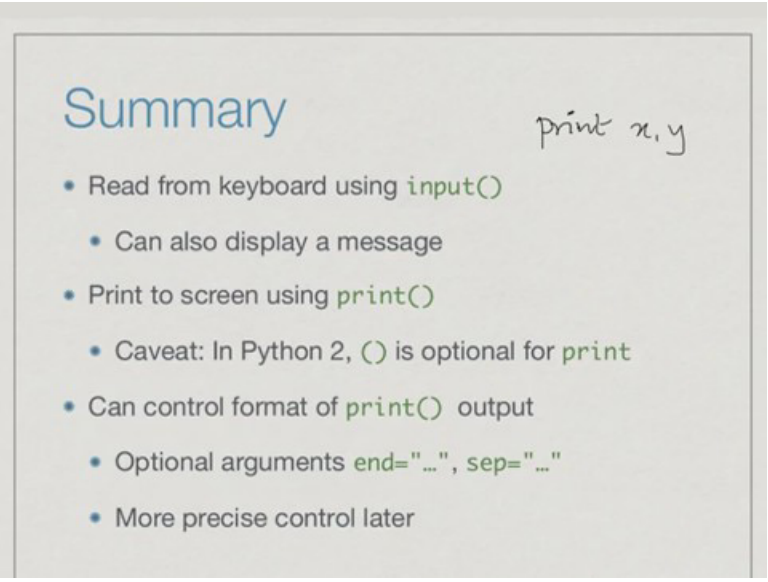


So, with the optional arguments end and sep, we can control when successive prints continue on the same line and we can control to some limited extent, how these values are separated on a line. But we may actually want to do a lot more. We might want to put a sequence of things, so that, they all line up, right.

Supposing, we want to print out a table using a print statement, we want to make sure that the columns line up. So, we might want to say that, each item that we want to print, like we have printing a sequence of numbers, line by line, because, the numbers may have different widths; some may be 3 digits, some may be 5 digits; we might say print them all to occupy 7 characters width, right. This is a thing that we might want to do, align text.

Now, within this alignment, we might want to align things left or right. If we have a default with, say 10 characters; if they are numbers, we might want them right aligned, so that the units digit is aligned up; if they are names, we might want them left aligned, so that we can read them from left to right, without it looking ragged. And if we are doing things like calculating averages or something, we may not want the entire thing to be displayed; we might want to truncate it to 2 decimal points; say, it is currency or something like that. These are all more intricate ways of formatting the output and we will see in the next lecture how to do this. But right now, you can use end and sep to do minimal formatting.

(Refer Slide Time: 14:53)



To summarize, you can use the input statement with an optional message, in order to read from the keyboard. You can print to the screen using the print statement. Now, we mentioned at the beginning that, there are some differences between python 2 and 3 and here is one of the more obvious differences that you will see, if you look at python 2 code, which is available from various sources.

In python 2, you can say print space and then, give the, what is given as the arguments to print in python 3. Python 3 insists on it being called like a function, with brackets; in python 2 the brackets are optional. So, you will very often see, in python 2 code, something that looks like print x, y, given without any brackets. This is legal in python 2; this is not legal in python 3. Just be careful about this.

And what we saw is that, with the limited amount of control, we can make print behave as we want. So, we can specify what to put at the end. In particular, we can tell it not to put a new line. So, continue printing in the same line and we can separate the values by something other than the default space character.