Loops

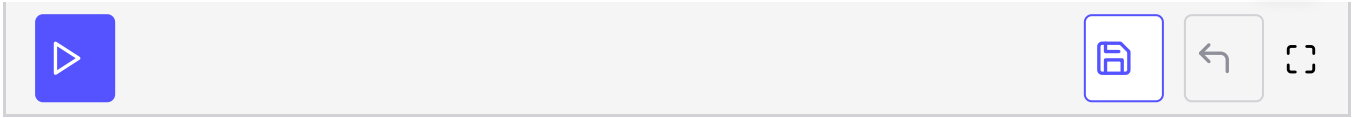
Introduction to Loops and their syntax in Python.

Computers are great for doing similar tasks many times — they don't mind, and they're very quick compared to humans with calculators!

Let's see if we can get a computer to print the first ten squared numbers, starting with 0 squared, 1 squared, then 2 squared and so on. We expect to see an output of something like 0, 1, 4, 9, 16, 25, and so on.

We could just do the calculation ourselves, and have a set of instructions like `print(0)`, `print(1)`, `print(4)`, and so on. This would work, but we would have failed to get the computer to do the calculation for us. More than that, we would have missed the opportunity to have a generic set of instruction to print the squares of numbers up to any specified value. To do this we need to pick up a few more new ideas, so we'll take it gently. Issue the following code into the next ready cell and run it.

```
print(list( range(10) ))
```



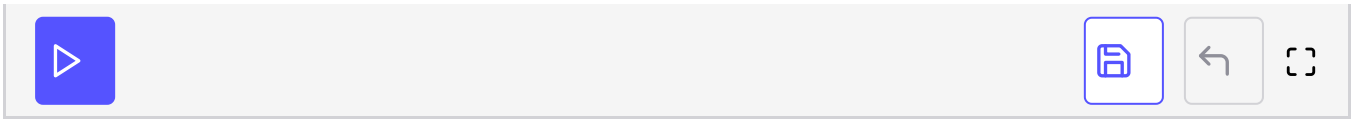
If you look at the output, you will see a list of ten numbers, from 0 up to 9. This is great because we got the computer to do the work to create the list, we didn't have to do it ourselves. We are the master, and the computer is our servant!

You may have been surprised that the list was from 0 to 9, and not from 1 to 10. This is because many computer related things start with 0 and not 1. It's tripped me up many times when I assumed a computer list started with 1 and not 0. Creating ordered lists are used to keep count when performing calculations, or indeed applying iterative functions, many times.

You may have noticed we missed out the “print” keyword, which we used when we printed the phrase `Hello World!`, but again didn’t when we evaluated 2*3. Using the “print” keyword can be optional when we’re working with Python in an interactive way because it knows we want to see the result of the instructions we issued.

A very common way to get computers to do things repeatedly is by using code structures called *loops*. The word loop does give you the right impression of something going round, and round potentially endlessly. Rather than define a loop, it’s easiest to see a simple one. Enter and run the following code in a new cell.

```
for n in range(10):
        print(n)
        pass
print('done')
```



There are three new things here so let’s go through them. The first line has the `range(10)` that we saw before. This creates a list of numbers from 0 to 9, as we saw before.

The “for n in” is the bit that creates a loop, and in this case, it does something for every number in the list and keeps count by assigning the current value to the variable n. We saw variables earlier, and this is just like assigning `n=0` during the first pass of the loop, then `n=1`, then `n=2`, until `n=9` which is the last item in the list.

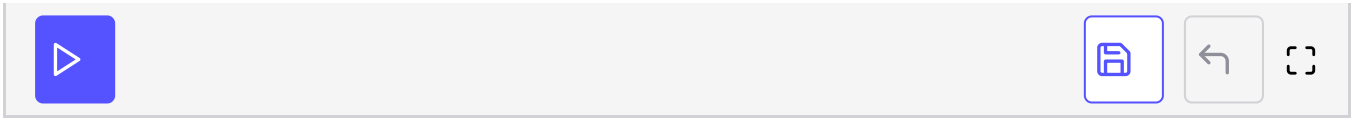
The next line `print(n)` shouldn’t surprise us by simply printing the value of *n*. We expect all the numbers in the list to be printed. But notice the indent before `print(n)`. This is important in Python as indents are used meaningfully to show which instructions are subservient to others, in this case, the loop created by “for n in …”. The “pass” instruction signals the end of the loop, and the next line is back at normal indentation and not part of the loop. This means we only expect “done” to be printed once, and not ten times.

It should be clear now that we can print the squares by printing `n*n`. In fact, we can make the output more helpful by printing phrases like “the square of 3

is 9". The following code shows this change to the print instruction repeated

inside the loop. Note how the variables are not inside quotes and are therefore evaluated.

```
for n in range(10):
    print("The square of", n, "is", n*n)
    pass
print('done')
```

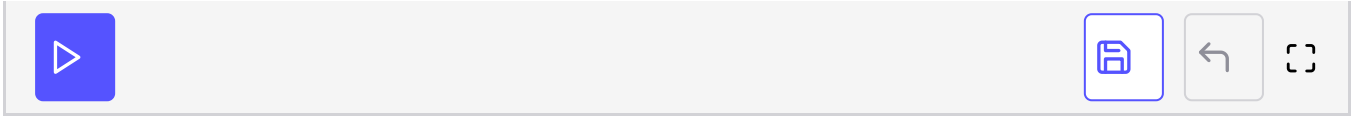


This is already quite powerful! We can get the computer to potentially do a lot of work very quickly with just a very short set of instructions. We could easily make the number of loop iterations much larger by using `range(50)` or even `range(1000)` if we wanted. Try it!

Comments

Before we discover more wild and wonderful Python commands, have a look at the following simple code and *Run* it.

```
# the following prints out the cube of 2
print(2**3)
```



The first line begins with a hash symbol `#`. Python ignores any lines beginning with a hash. Rather than being useless, we can use such lines to place helpful comments into the code to make it clearer for other readers, or even ourselves if we came back to the code at a later time.

Trust me. You will be thankful you commented your code, especially the more complex or less obvious bits of code. The number of times I have tried to decode my own code, asking myself "what was I thinking ..."