How to Read a File

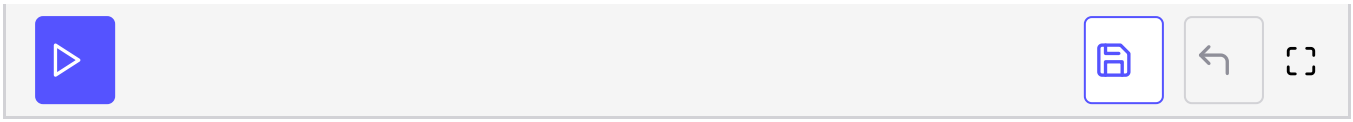
Let's learn how to read a file in python

Python has a builtin function called **open** that we can use to open a file for reading. For our examples in this course, we will use a text file name “test.txt” with the following contents (don’t worry, we’ve already worked with Educative to have these files uploaded to your execution directory):

```
This is a test file
line 2
line 3
this line intentionally left blank
```

Here are a couple of examples that show how to use **open** for reading:

```
handle = open("test.txt")
handle = open(r"py101book/data/test.txt", "r")
```



The first example will open a file named **test.txt** in read-only mode. This is the default mode of the **open** function. Note that we didn’t pass a fully qualified path to the file that we wanted to open in the first example. Python will automatically look in the folder that the script is running in for **test.txt**. If it doesn’t find it, then you will receive an IOError.

The second example does show a fully qualified path to the file. The above code would run fine on linux machines but what if you want to give a windows style paths. This is how it’s done

```
handle = open(r"py101book\data\test.txt", "r")
```

If you look at the path, you’ll notice that it begins with an “r”. This means that

we want Python to treat the string as a raw string. Let's take a moment to see

the difference between specifying a raw string versus a regular string:

```
print("py101book\data\test.txt")
# py101book\data        est.txt

print(r"py101book\data\test.txt")
# py101book\data\test.txtx
```

As you can see, when we don't specify it as a raw string, we get an invalid path. Why does this happen? Well, as you might recall from the strings chapter, there are certain special characters that need to be escaped, such as "n" or "t". In this case, we see there's a "t" (i.e. a tab), so the string obediently adds a tab to our path and screws it up for us.

The second argument in the second example is also an "r". This tells **open** that we want to open the file in read-only mode. In other words, it does the same thing as the first example, but it's more explicit. Now let's actually read the file!

Put the following lines into a Python script and save it in the same location as your test.txt file:

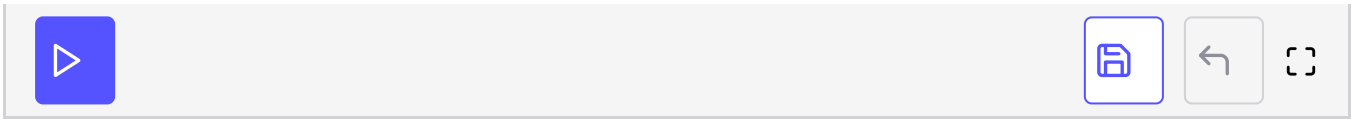
```
handle = open("test.txt", "r")
data = handle.read()
print(data)
handle.close()
```

If you run this, it will open the file and read the entire file as a string into the **data** variable. Then we print that data and close the file handle. You should always close a file handle as you never know when another program will want to access it. Closing the file will also help save memory and prevent strange bugs in your programs. You can tell Python to just read a line at a time, to read all the lines into a Python list or to read the file in chunks. The last option is very handy when you are dealing with really large files and you

don't want to read the whole thing in, which might fill up the PC's memory.

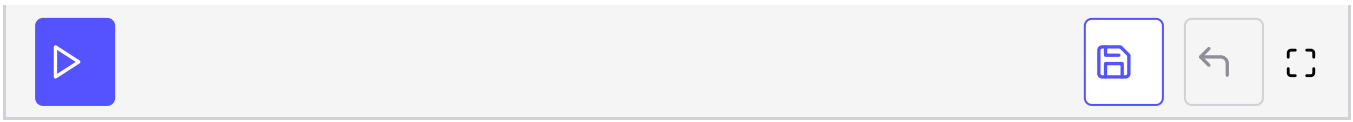
Let's spend some time looking at different ways to read files.

```
handle = open("test.txt", "r")
data = handle.readline() # read just one line
print(data)
handle.close()
```



If you run this example, it will only read the first line of your text file and print it out. That's not too useful, so let's try the file handle's readlines() method:

```
handle = open("test.txt", "r")
data = handle.readlines() # read ALL the lines!
print(data)
handle.close()
```



After running this code, you will see a Python list printed to the screen because that's what the **readlines** method returns: a list! Let's take a moment to learn how to read a file in smaller chunks.