The Popen Class

How python's Popen works to create a child process?

The **Popen** class executes a child program in a new process. Unlike the **call** method, it does not wait for the called process to end unless you tell it to using by using the **wait** method. Let's try running **du** through Popen and see if we can detect a difference:

```
import subprocess
subprocess.Popen("du")
```

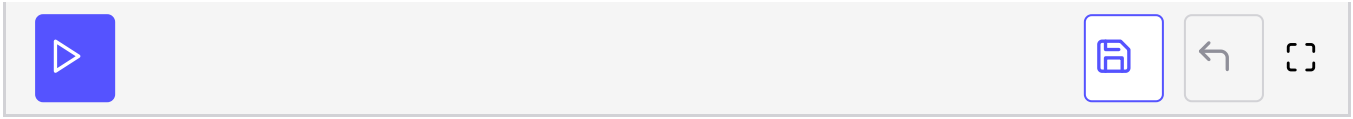
Here we create a variable called **program** and assign it the value of "notepad.exe". Then we pass it to the Popen class. When you run this, you will see that it immediately returns the **subprocess.Popen object** and the application that was called executes. Let's make Popen wait for the program to finish:

```
import subprocess
program = "du"
process = subprocess.Popen(program)
code = process.wait()
print(code)
```

Note that using the **wait** method can cause the child process to deadlock when using the stdout/stderr=PIPE commands when the process generates enough output to block the pipe. You can use the **communicate** method to alleviate this situation. We'll be looking at that method in the next section.

Now let's try running Popen using multiple arguments:

```
import subprocess
subprocess.Popen(["ls", "-l"])
#<subprocess.Popen object at 0xb7451001>
```



If you run this code in Linux, you'll see it print out the Popen object message and then a listing of the permissions and contents of whatever folder you ran this in. You can use the shell argument with Popen too, but the same caveats apply to Popen as they did to the call method.