Learning to Communicate with subprocess

How to communicate with the subprocess in Python

WE'LL COVER THE FOLLOWING

- Wrapping Up

There are several ways to communicate with the process you have invoked. We're just going to focus on how to use the subprocess module's **communicate** method. Let's take a look:

```
import subprocess

args = ["du"]
process = subprocess.Popen(args,
                           stdout=subprocess.PIPE)

data = process.communicate()
print(data)
```

In this code example, we create an **args** variable to hold our list of arguments. Then we redirect standard out (stdout) to our subprocess so we can communicate with it. The **communicate** method itself allows us to communicate with the process we just spawned. We can actually pass input to the process using this method. But in this example, we just use communicate to read from standard out. You will notice when you run this code that communicate will wait for the process to finish and then returns a two-element tuple that contains what was in stdout and stderr. Here is the result from my run:

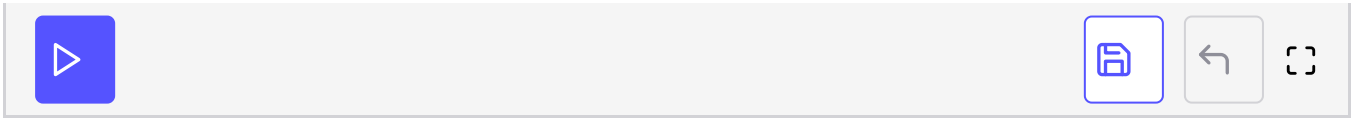
```
(b'4\t./output\n8\t./py101book/data\n12\t./py101book\n56\t.\n', None)
```

That's kind of ugly. Let's make it print out the result in a more readable format, shall we?

```
import subprocess

args = ["du"]
process = subprocess.Popen(args, stdout=subprocess.PIPE)

data = process.communicate()
for line in data:
    print(line)
```



If you run this code, you should see something like the following printed to your screen:

```
b'4\t./output\n8\t./py101book/data\n12\t./py101book\n56\t.\n'
None
```

That last line that says "None" is the result of stderr, which means that there were no errors.

Wrapping Up

At this point, you have the knowledge to use the subprocess module effectively. You can open a process in two different ways, you know how to wait for a return code, and you know how to communicate with the child process that you created.

In the next chapter, we will be looking at the **sys** module.