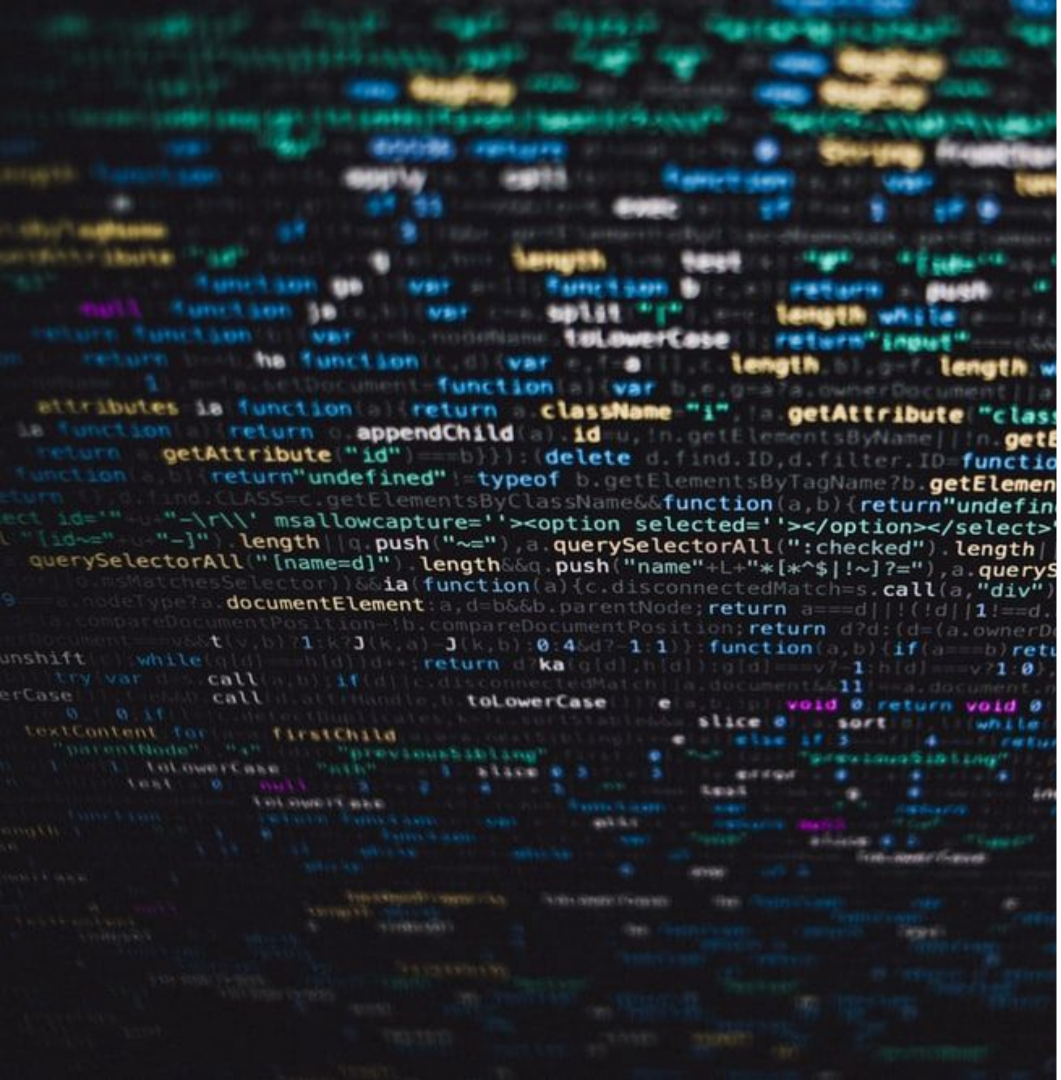
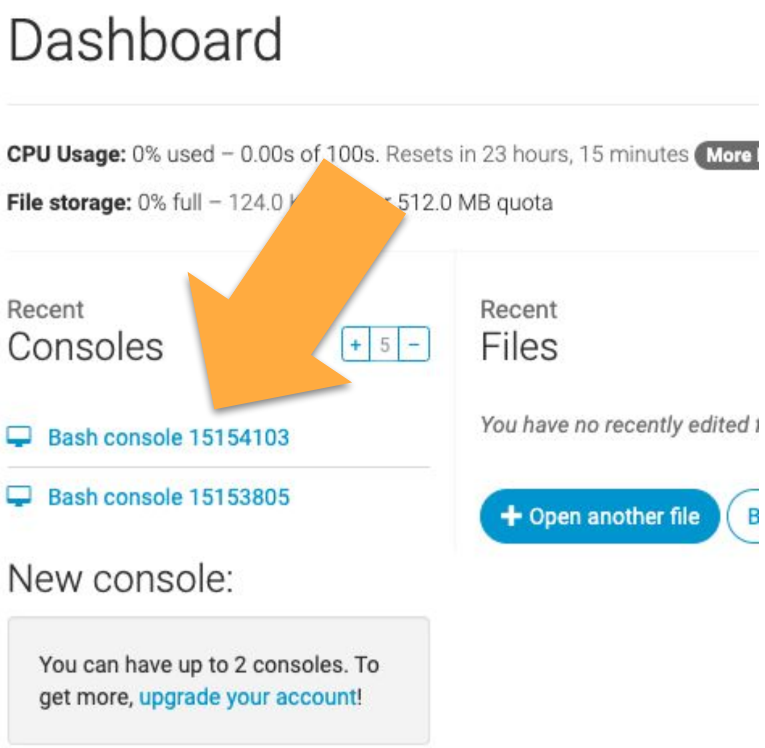


CODE CAMP

Day 002: Input and Decisions


Dashboard

CPU Usage: 0% used – 0.00s of 100s. Resets in 23 hours, 15 minutes More

File storage: 0% full – 124.0 512.0 MB quota

Recent Consoles

Bash console 15154103

Bash console 15153805

New console:

You can have up to 2 consoles. To get more, upgrade your account!

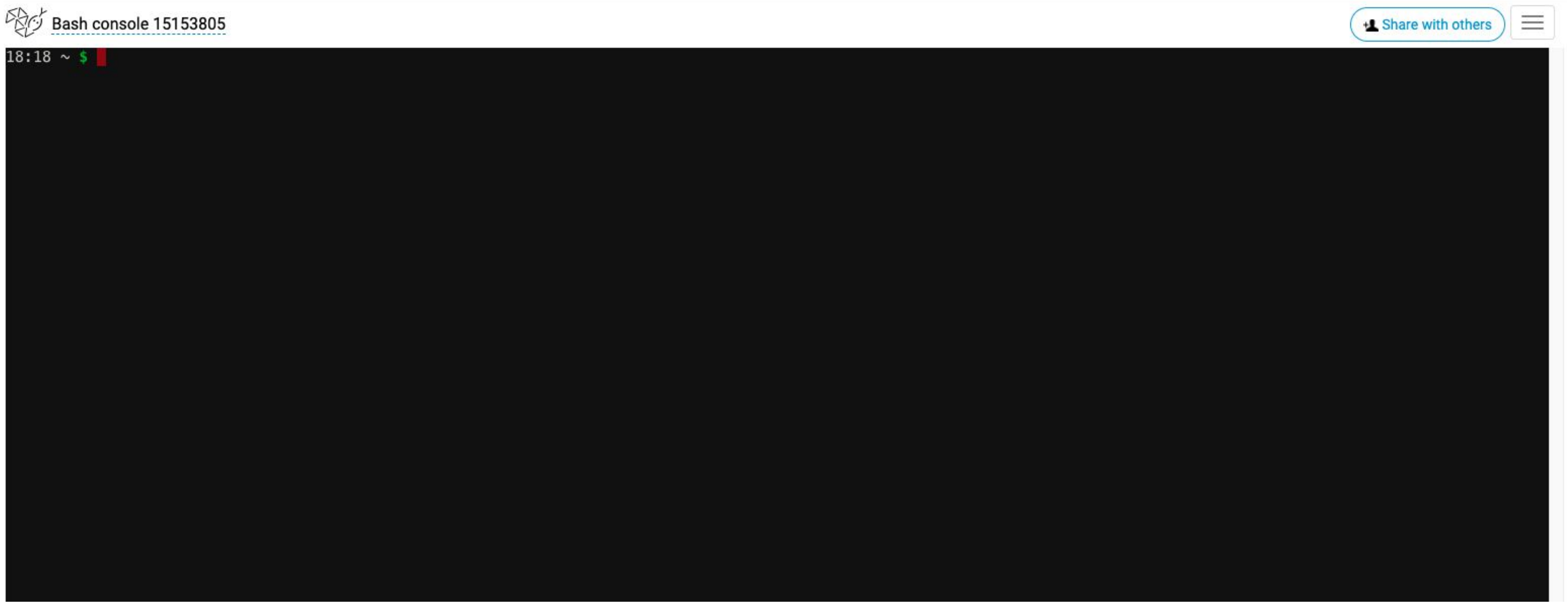
Recent Files

You have no recently edited

.001 Log In to Python Anywhere

Visit PythonAnywhere.com and log in to your account. You can click on one of the bash consoles you logged in to previously. They are listed under “Recent Consoles.”

If your command line is cluttered from your previous work you can type the **clear** command to clear the console visual history.



```
18:18 ~ $
```

.001 Log In to Python Anywhere

If you've done everything correctly, your screen should appear similar to the image above. At your command prompt, type the command **nano** to open your text editor.

.002 Obtaining Command Line Input

One way you can pass input to a python script is through the command line. When you run the program you typically use the command **python** and then the filename of the script.

When you pass a parameter to the python script the command format looks like this:

```
python <filename.py> <parameter>
```

So if your files was called “keys.py” and your parameter was “xoxo” the command line input would be:

```
python keys.py xoxo
```

```
import sys

print("Parameter:  " + (sys.argv[1]) )
```

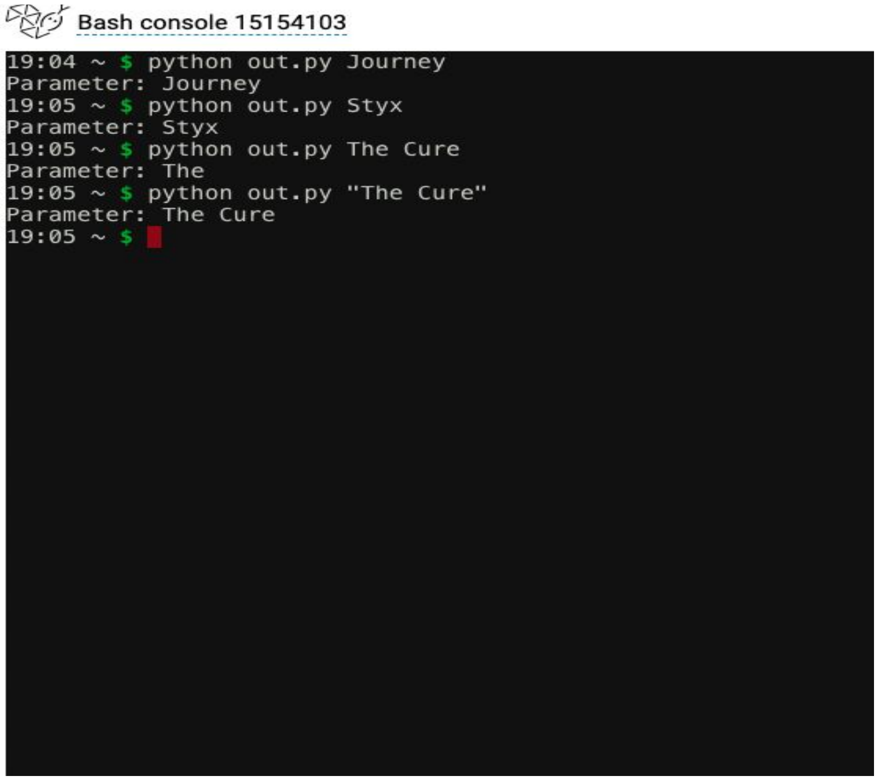
.002 Obtaining Command Line Input

Type the code at the left in to your nano buffer. (Each file in nano lives in its on “buffer.”)

Be especially careful with the syntax in the second line.

When you’ve double-checked your code, save the file using the **Write Out** control key in Nano. Use the file name **out.py**.

Exit Nano and return to your command line prompt.


Bash console 15154103

```
19:04 ~ $ python out.py Journey
Parameter: Journey
19:05 ~ $ python out.py Styx
Parameter: Styx
19:05 ~ $ python out.py The Cure
Parameter: The
19:05 ~ $ python out.py "The Cure"
Parameter: The Cure
19:05 ~ $
```

.002 Obtaining Command Line Input

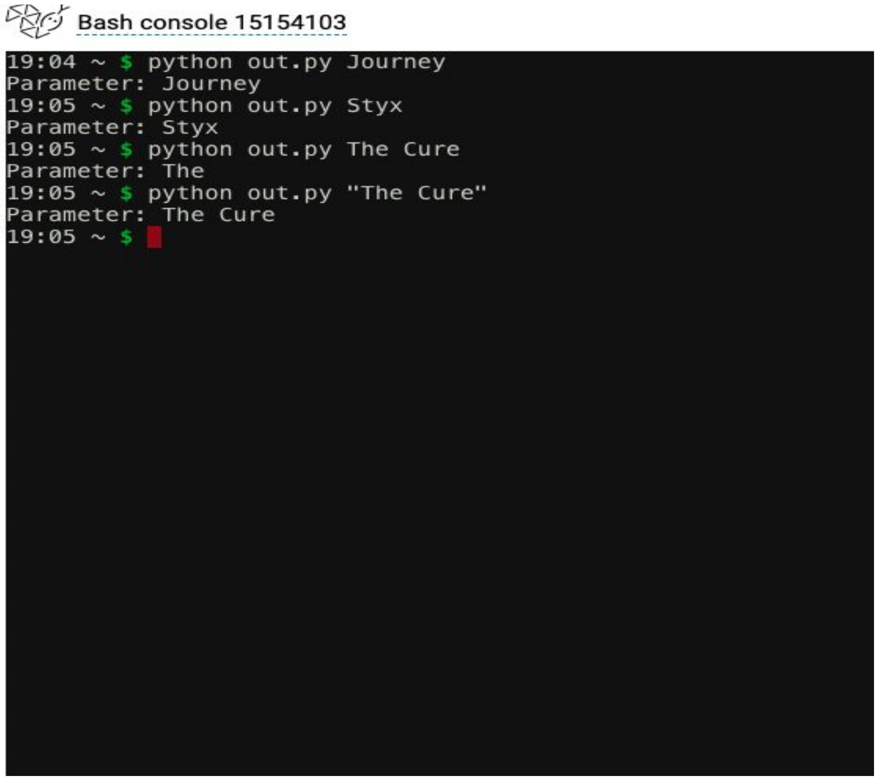
Let’s test our short program by passing a number of different values to the command line. For example try the following:

python out.py chocolate

You should see the response:
Parameter: Chocolate

If you do not get the correct response, reload your program in Nano by typing **nano out.py**. Correct any errors that you see.

Note that if you pass more than one word to the Python script the parameter must be enclosed in quotes.

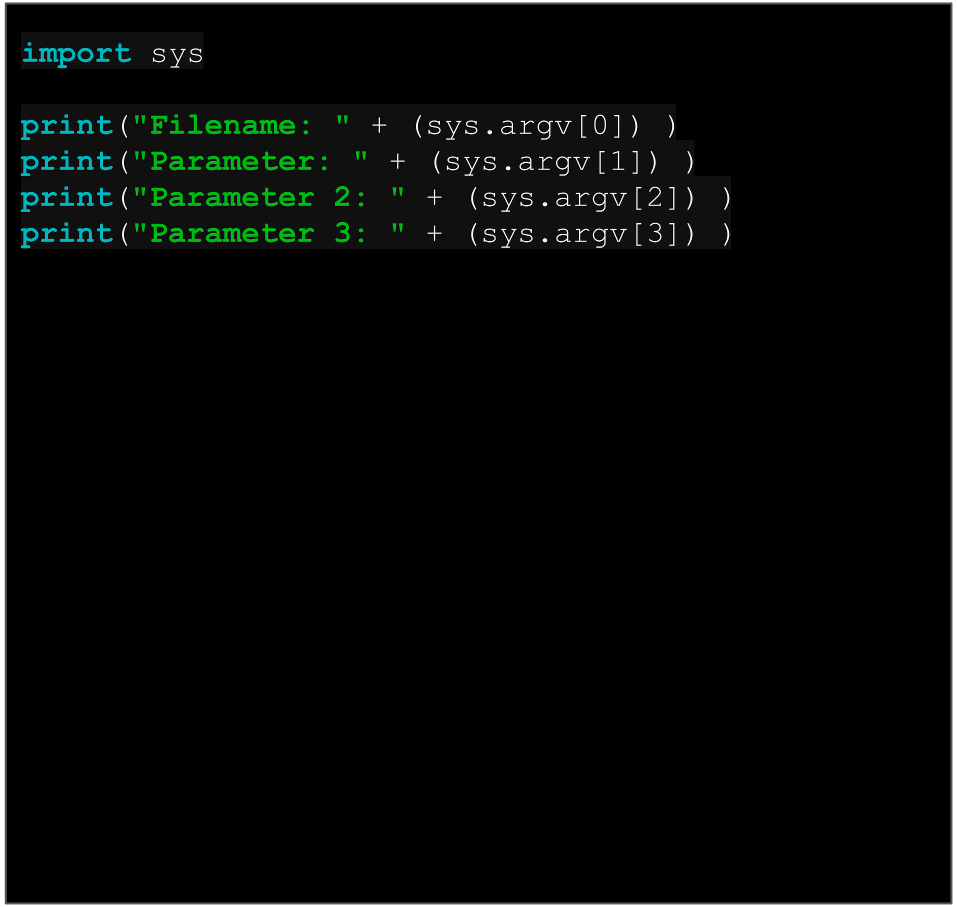


```
19:04 ~ $ python out.py Journey
Parameter: Journey
19:05 ~ $ python out.py Styx
Parameter: Styx
19:05 ~ $ python out.py The Cure
Parameter: The
19:05 ~ $ python out.py "The Cure"
Parameter: The Cure
19:05 ~ $
```

.002 Obtaining Command Line Input

Try passing a few different values to your Python script through the command line to make sure everything works as expected.

You may want to try numerical values as well as string values.



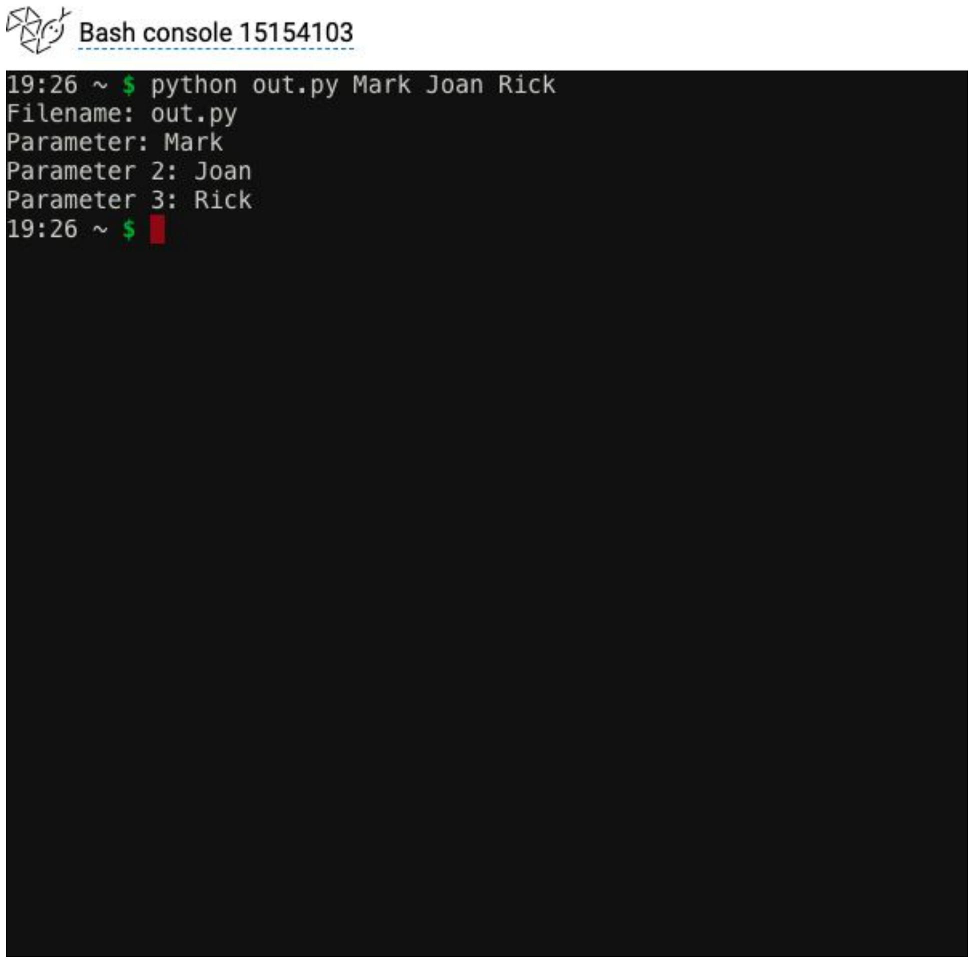
```
import sys

print("Filename: " + (sys.argv[0]) )
print("Parameter: " + (sys.argv[1]) )
print("Parameter 2: " + (sys.argv[2]) )
print("Parameter 3: " + (sys.argv[3]) )
```

.002 Obtaining Command Line Input

Reload your script in nano by entering **nano out.py** in to the command line.

Alter your code to add the lines appearing on the left.

Save your file and exit nano.



Bash console 15154103

```
19:26 ~ $ python out.py Mark Joan Rick
Filename: out.py
Parameter: Mark
Parameter 2: Joan
Parameter 3: Rick
19:26 ~ $
```

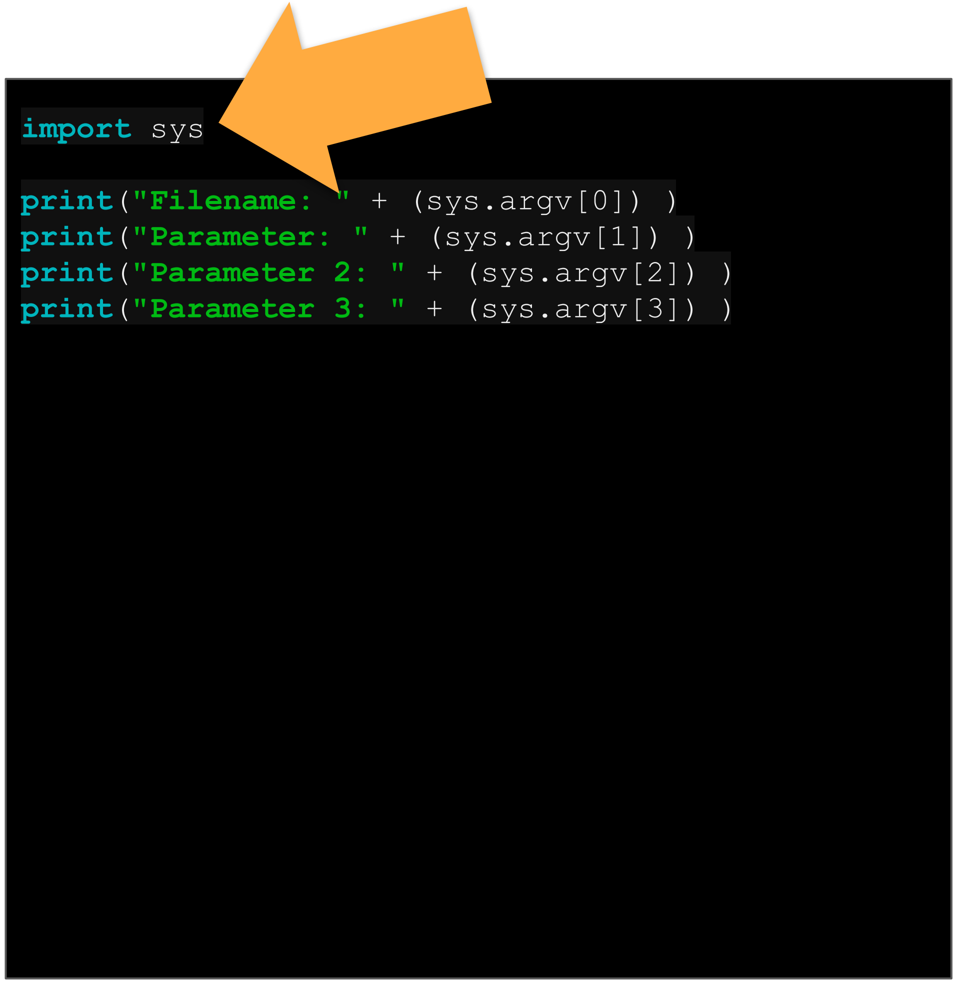
.002 Obtaining Command Line Input

Trying running the program and passing three parameters-- the names of three family members.

The format should be:

python out.py <name1> <name2> <name3>

If everything is working as expected, move on. If not, debug your program until you get the correct result.

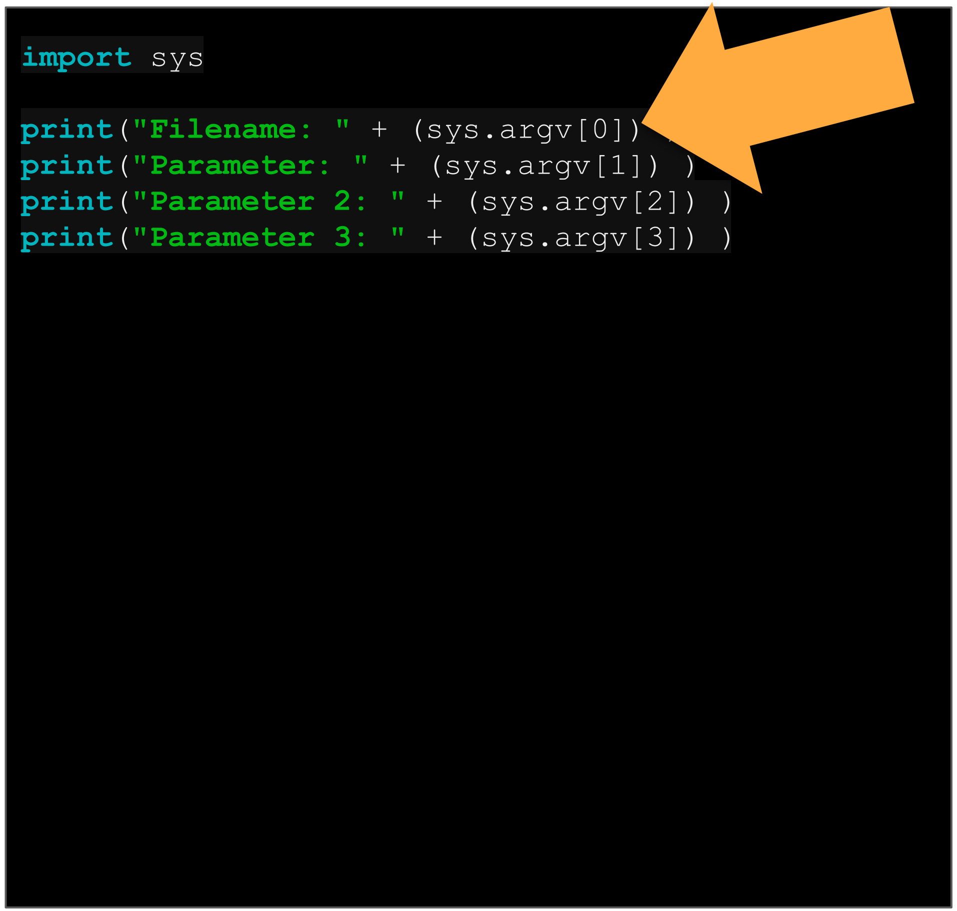


```
import sys

print("Filename: " + (sys.argv[0]) )
print("Parameter: " + (sys.argv[1]) )
print("Parameter 2: " + (sys.argv[2]) )
print("Parameter 3: " + (sys.argv[3]) )
```

.003 Analyzing the Code

Organizationally, Python stores code in packages. There are dozens (hundreds?) of Python packages that expand the core functionality of Python.

To access command line arguments we need the system package which is abbreviated **sys**. The import command allows us to use any Python command in a designated package.



```
import sys

print("Filename: " + (sys.argv[0])
print("Parameter: " + (sys.argv[1]) )
print("Parameter 2: " + (sys.argv[2]) )
print("Parameter 3: " + (sys.argv[3]) )
```

.003 Analyzing the Code

The arguments sent with a **python** command are accessible through an array. (An array is several values stored together, as a set). We'll learn more about arrays later on during the code camp. If we used typed in to the command line:

python out.py Brett Kerry Colleen

"out.py" would occupy position 0 in the array. "Brett" would occupy position 1. "Kerry" would occupy position 2 and "Colleen" position 3. If you try to access a parameter number that isn't supplied, an error will result. Try sending only one or two parameters to our current program.

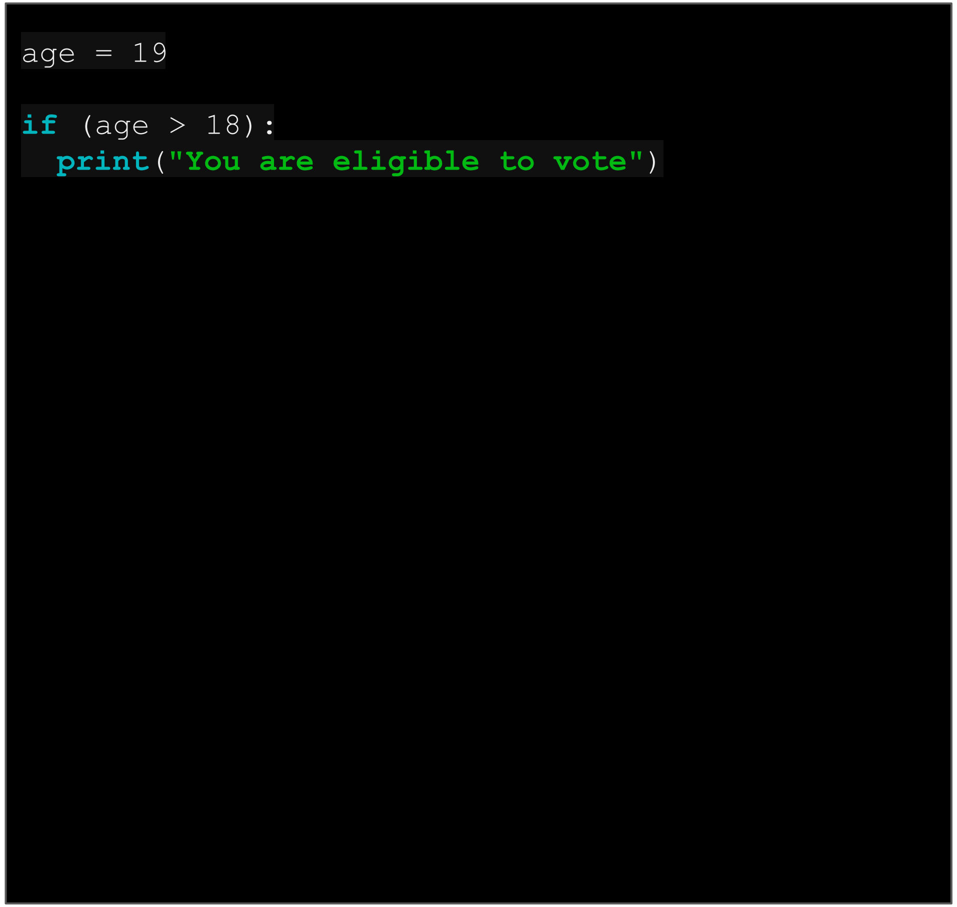
.004 Conditional Statements

Conditional statements allow your code to make decisions. Decisions are made based on a logical statement that is evaluated as either true or false.

A logical statement, in plain English, might be “is equal to 15”, or “is greater than 100.”

In code the logical statement would look like this:

x > 15	true if x is greater than 15
x == 85	true if x is equal to 85
x >= 18	true if x is greater than or equal to 18
x **== “Mark”**	true if the value of x is “Mark”



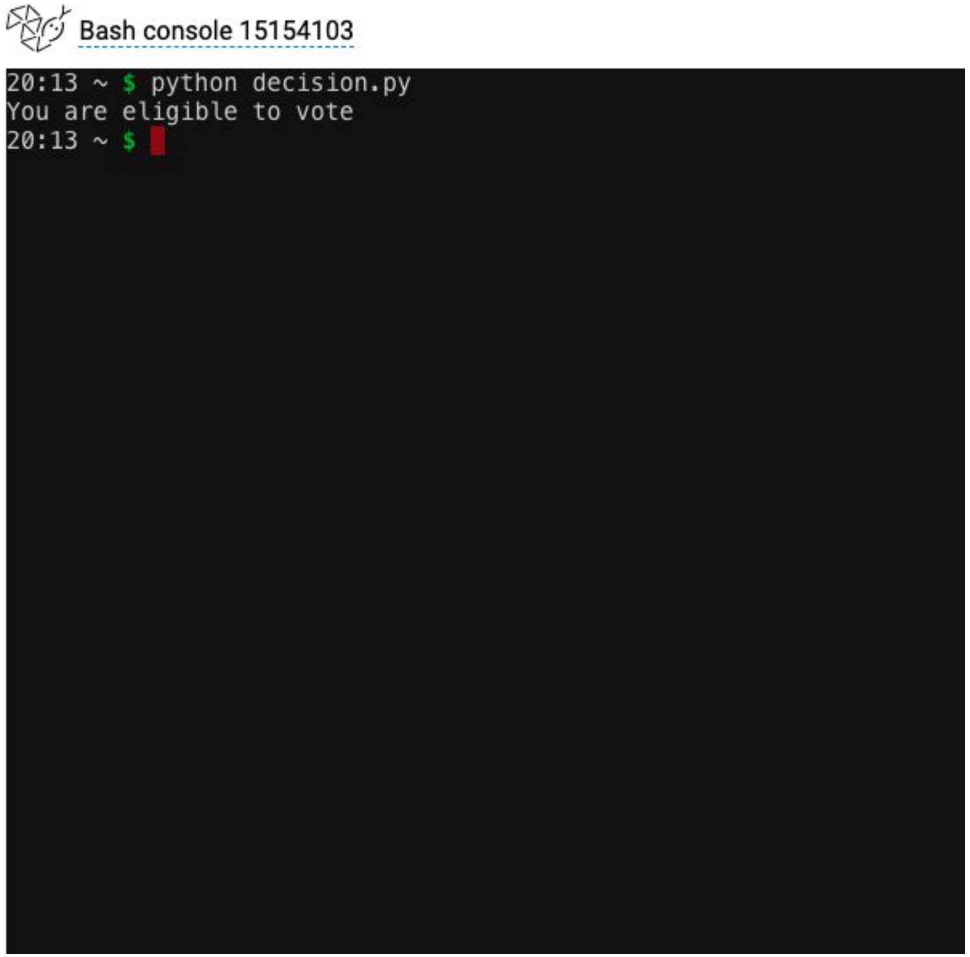
```
age = 19

if (age > 18):
  print("You are eligible to vote")
```

.004 Conditional Statements

Create a new Nano buffer and enter the code on the left and save as **decision.py**.

Exit **nano**. And use the **python** command to execute your code. The condition here we're testing is **age > 18**.

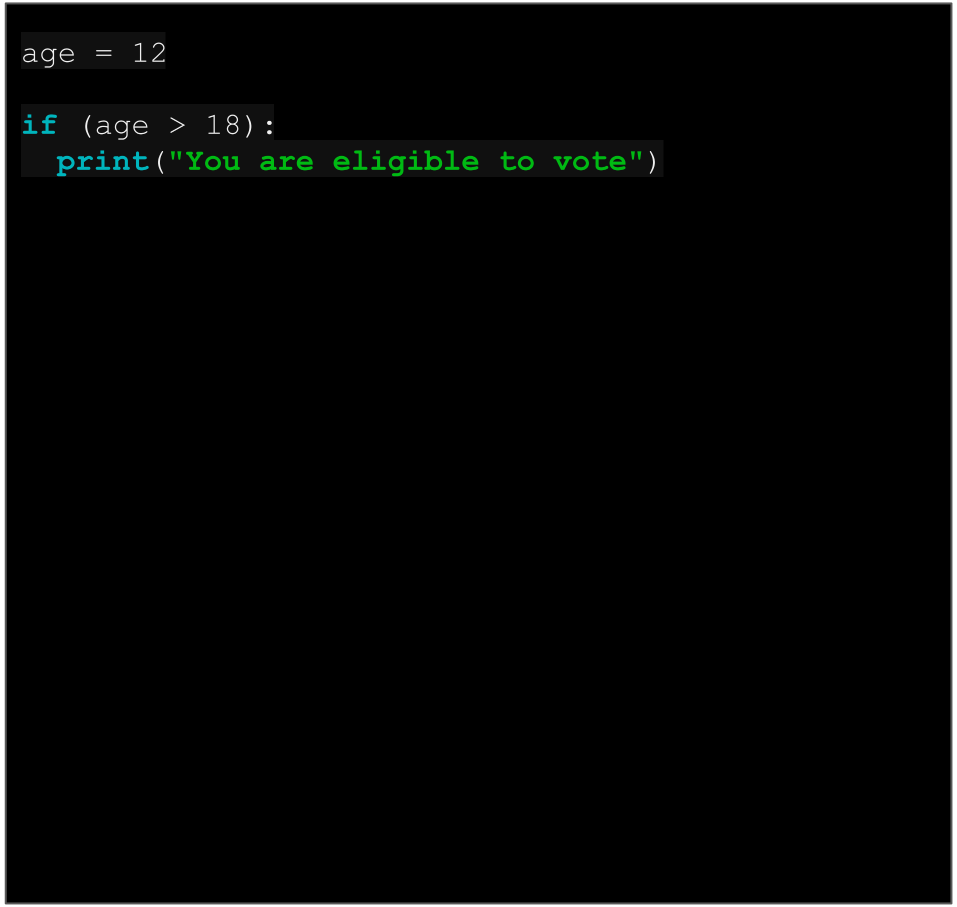
In this case, the statement should evaluate as true because the value of the variable **age** is 19-- which, of course, is greater than 18.


Bash console 15154103

```
20:13 ~ $ python decision.py
You are eligible to vote
20:13 ~ $ 
```

.004 Conditional Statements

When executing the Python script our statement was evaluated as true and the expected output is shown.

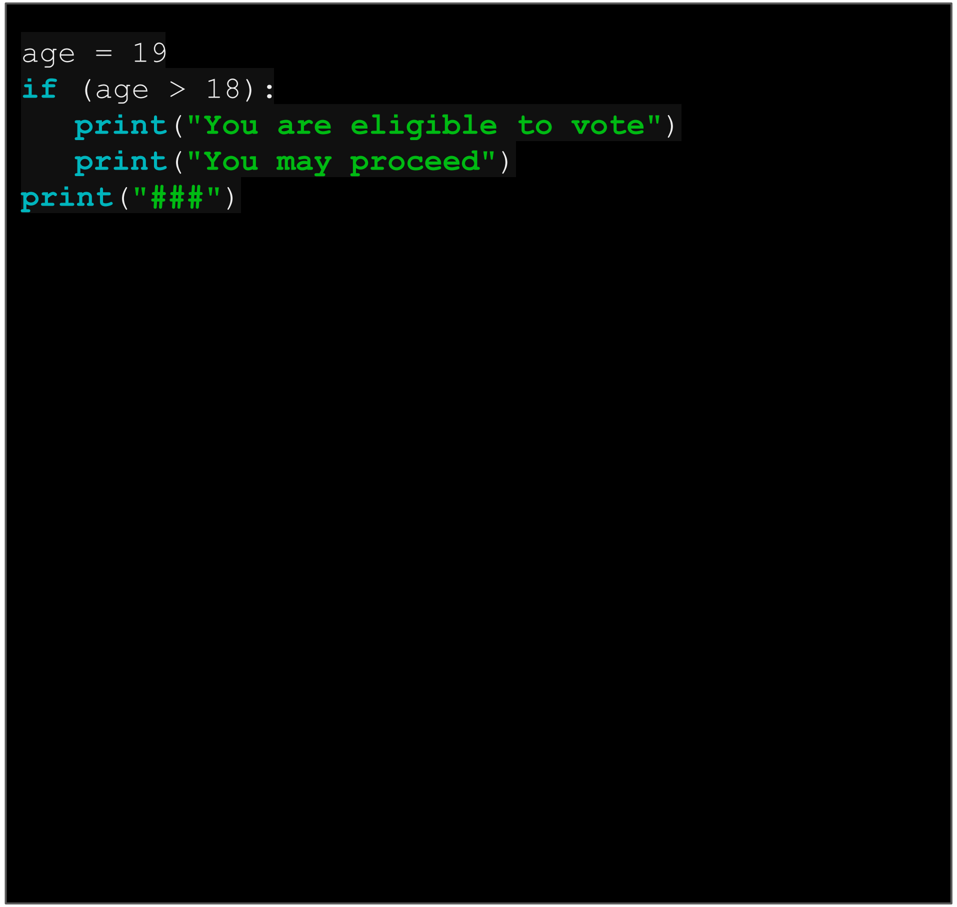


```
age = 12

if (age > 18):
  print("You are eligible to vote")
```

.004 Conditional Statements

Let's make a quick change to our program. Change the value of the variable **age** to **12**.

Save the program and execute the script on the command line. If you have done everything correct there will be no actual output from the program, because the initial condition is false and the indented code underneath the **if** statement does not execute.



```
age = 19
if (age > 18):
   print("You are eligible to vote")
   print("You may proceed")
print("###")
```

.004 Conditional Statements

Let’s change our program yet again.

Save the program and execute the script on the command line. Note that indentation is important in Python. If your conditional statement is true, every line of code indented under the **if** statement is executed. If the conditional statement is false every line of code under the conditional statement is ignored.

In this case the final line of code which prints “###” is executed regardless of whether or not the if statement is resolved as true.

```
import sys

age = int(sys.argv[1])
if (age >= 18):
   print("You are eligible to vote")
   print("You may proceed")
else:
   print("You are not eligible to vote")
```

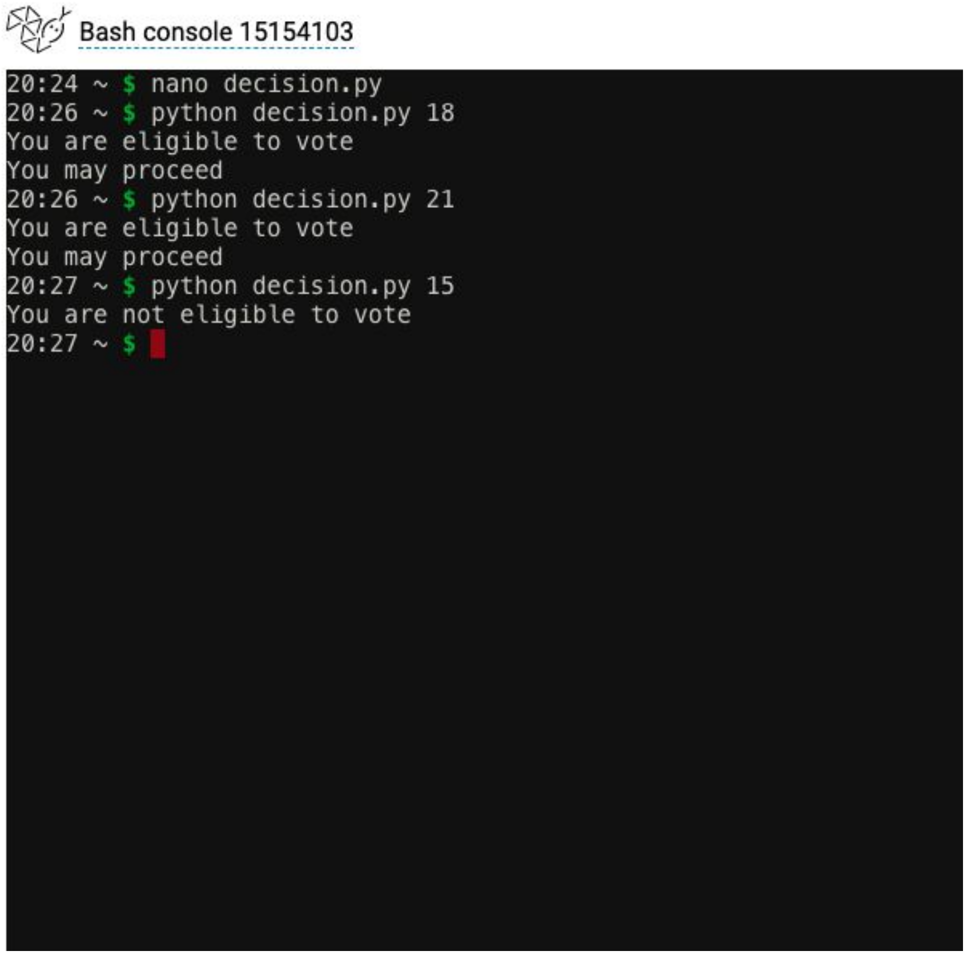
.004 Conditional Statements

Let’s update our program. Enter the code on the left.

Without running the program, from the code, can you anticipate what it will do?

Save your code and exit Nano and return to your command line. Issue the following command:

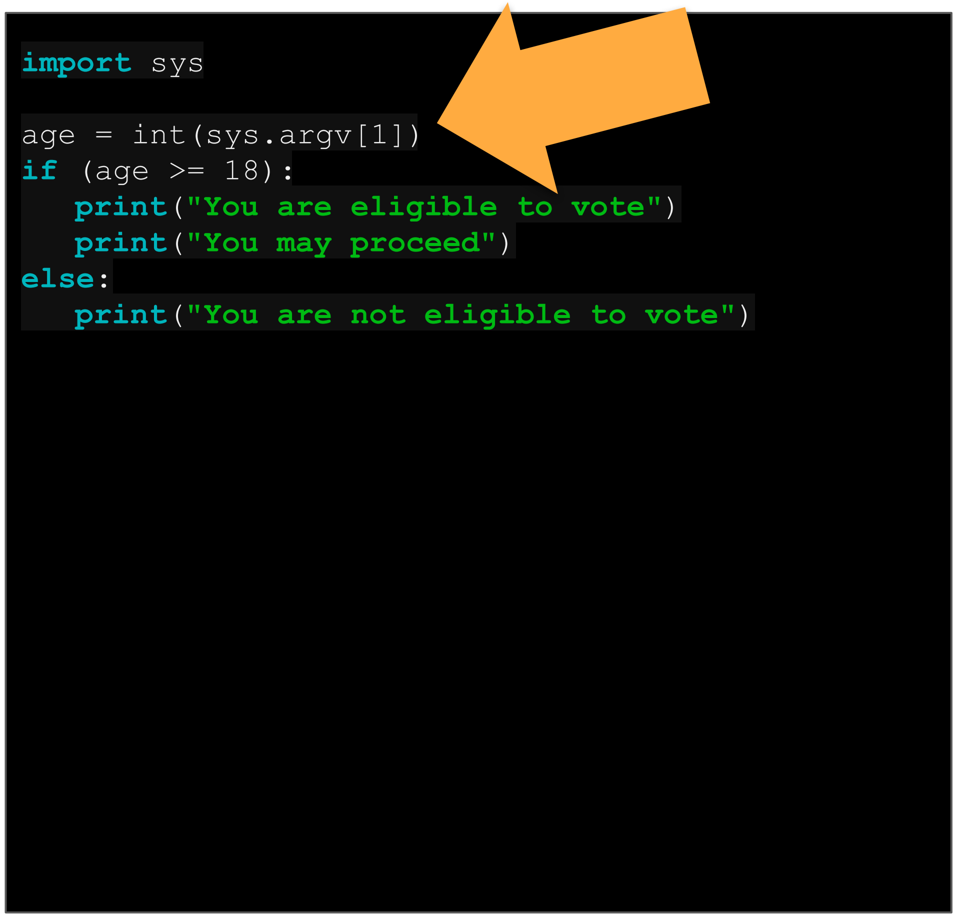
python decision.py 21


Bash console 15154103

```
20:24 ~ $ nano decision.py
20:26 ~ $ python decision.py 18
You are eligible to vote
You may proceed
20:26 ~ $ python decision.py 21
You are eligible to vote
You may proceed
20:27 ~ $ python decision.py 15
You are not eligible to vote
20:27 ~ $
```

.004 Conditional Statements

Run a few additional tests and ensure your program runs correctly, using the screenshot on the left as your guide.

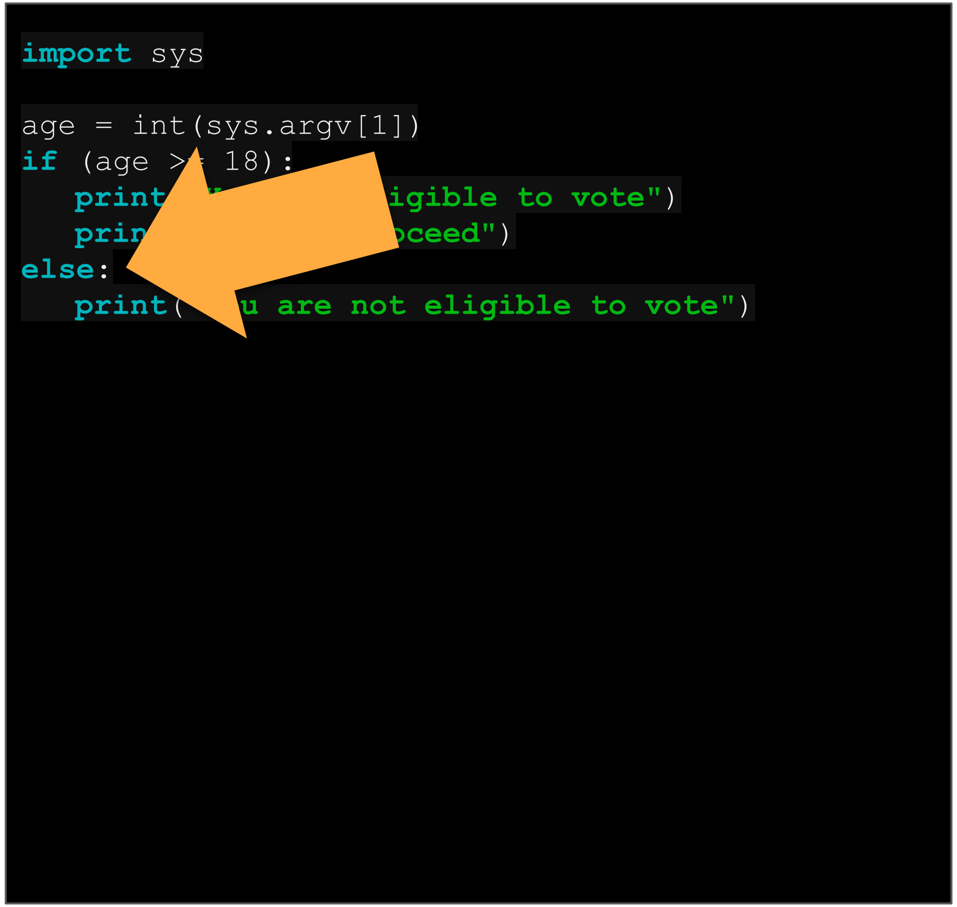


```
import sys

age = int(sys.argv[1])
if (age >= 18):
   print("You are eligible to vote")
   print("You may proceed")
else:
   print("You are not eligible to vote")
```

.005 Analyzing Your Code

This program takes the first command line parameter provided after the filename and assigns it to the variable **age**.

Before assigning the value to the **age** variable the **int** function is run. The **int** function converts the **string** input to an integer value. It must be converted because the logical operators used with the **if** statement (==, >, etc.) only work with numerical values, not strings.



```
import sys

age = int(sys.argv[1])
if (age >= 18):
    print("        igible to vote")
    prin          oceed")
else:
    print(   u are not eligible to vote")
```

.005 Analyzing Your Code

The **else** statement runs the code indented below it if the initial condition is evaluated as false.

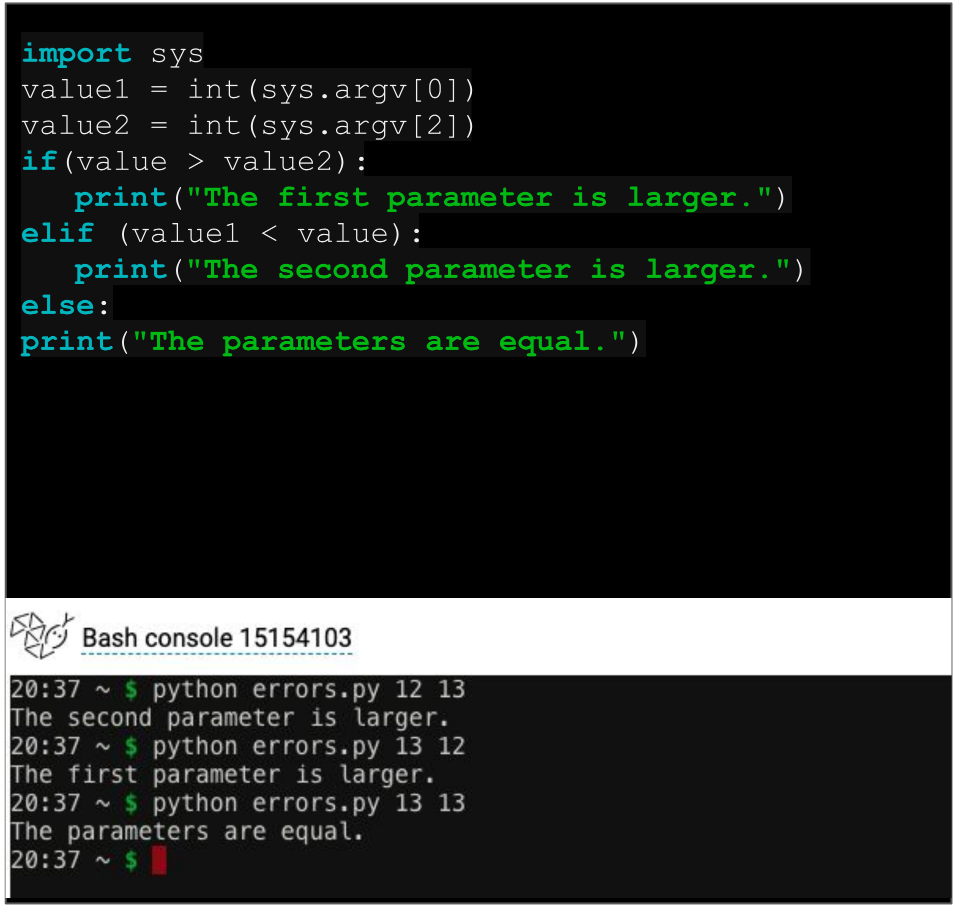
If you were to enter:

Python decision.py 11

The result would be:

You are not eligible to vote.

The **else** statement would run because when the comparison is made **11 >= 18** would evaluate as **false.**



```
import sys
value1 = int(sys.argv[0])
value2 = int(sys.argv[2])
if(value > value2):
   print("The first parameter is larger.")
elif (value1 < value):
   print("The second parameter is larger.")
else:
print("The parameters are equal.")
```

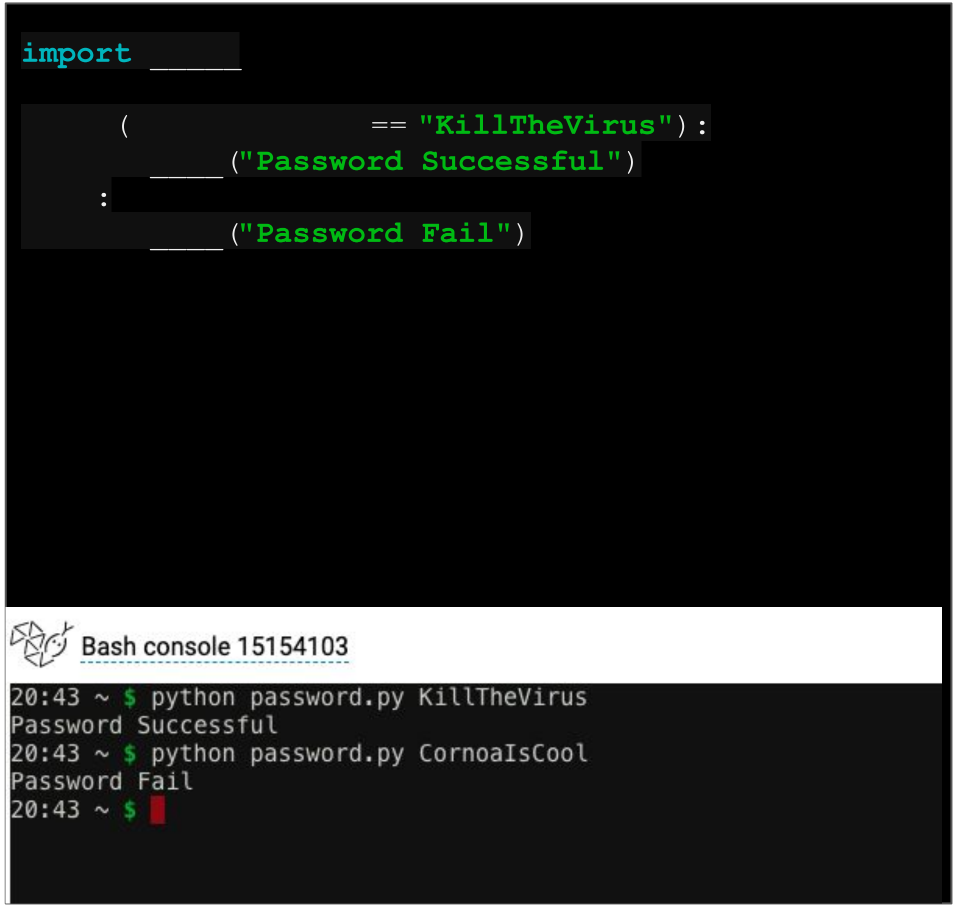
Bash console 15154103

```
20:37 ~ $ python errors.py 12 13
The second parameter is larger.
20:37 ~ $ python errors.py 13 12
The first parameter is larger.
20:37 ~ $ python errors.py 13 13
The parameters are equal.
20:37 ~ $
```

.006 Activity 001

Consider the bash console output at the bottom of the screen. With that output in mind, correct the code on the left so that it generates the correct output without errors.

Hint: **elif** is the correct way to combine an **if** statement and an **else** statement. The **elif** statement has its own logical statement which is evaluated as true or false.



```
import ______

      (                  == "KillTheVirus"):
          _____("Password Successful")
     :
          _____("Password Fail")
```

Bash console 15154103

```
20:43 ~ $ python password.py KillTheVirus
Password Successful
20:43 ~ $ python password.py CornoaIsCool
Password Fail
20:43 ~ $
```

.006 Activity 002

Consider the bash console output at the bottom of the screen. With that output in mind, fill in the blanks in the code so that the program functions as demonstrated.

The Ten Commandments For Computer Ethics

from the Computer Ethics Institute

1. Thou shalt not use a computer to harm other people.
2. Thou shalt not interfere with other people's computer work.
3. Thou shalt not snoop around in other people's files.
4. Thou shalt not use a computer to steal.
5. Thou shalt not use a computer to bear false witness.
6. Thou shalt not use or copy software for which you have not paid.
7. Thou shalt not use other people's computer resources without authorization.
8. Thou shalt not appropriate other people's intellectual output.
9. Thou shalt think about the social consequences of the program you write.
10. Thou shalt use a computer in ways that show consideration and respect.

.006 Activity 003

Review the Computer Ethics Institute Ten Commandments on the left. Pick one or two and discuss why you think they are important.