

Assignment Operators

Augmented assignment statements

Augmented assignment is the combination, in a single statement, of a binary operation and an assignment statement:

Operators	Description
<code>+=</code>	<pre>A=5 A=A+1 → A+=1 B=6 → B=B+5 → B+=5 X=10 Y=5 X+=Y → X=X+Y</pre>
<code>-=</code>	<pre>A=5 A=A-2 → A-=2 >>> s1="Python" >>> s1+="Language" >>> print(s1) PythonLanguage >>> n1=5 >>> n2=6 >>> n1-=n2 >>> print(n1) -1</pre>
<code>*=</code>	<pre>>>> l1=[1,2,3,4,5] >>> l1=l1*2 >>> print(l1) [1, 2, 3, 4, 5, 1, 2, 3, 4, 5] >>> l2=[10,20,30,40,50] >>> l2*=2 >>> print(l2) [10, 20, 30, 40, 50, 10, 20, 30, 40, 50] >>> n1=5 >>> n2=3 >>> n1*=n2 >>> print(n1,n2) 15 3 >>> a=10 >>> b=5 >>> c=a+=b SyntaxError: invalid syntax</pre>
<code>/=</code>	<pre>>>> n1=5</pre>

	<pre> >>> n1=n1/2 >>> print(n1) 2.5 >>> n2=5 >>> n2/=2 >>> print(n2) 2.5 </pre>
//=	<pre> >>> n1=5 >>> n1=n1//2 >>> print(n1) 2 >>> n2=5 >>> n2//=2 >>> print(n2) 2 </pre>
%=	<pre> >>> num=8 >>> num%=3 >>> print(num) 2 </pre>
=	<pre> >>> n1=5 >>> n1=2 >>> print(n1) 25 >>> n2=5 >>> n3=3 >>> n2**=n3 >>> print(n2) 125 </pre>
&=	<pre> >>> a=0b101 >>> b=0b111 >>> a=a&b >>> print(bin(a),bin(b)) 0b101 0b111 >>> print(a,b) 5 7 >>> n1=0b101 >>> n2=0b100 >>> n1&=n2 >>> print(bin(n1),bin(n2)) 0b100 0b100 </pre>

=	<pre> >>> x=0b101 >>> y=0b100 >>> x=x y print(bin(x),bin(y)) 0b101 0b100 x =y print(bin(x),bin(y)) 0b101 0b100 </pre>
^=	<pre> >>> n1=0b101 >>> n2=0b111 >>> n1^=n2 >>> print(bin(n1)) 0b10 </pre>
>>=	<pre> >>> x=0b1010 >>> x>>=2 >>> print(bin(x)) 0b10 >>> print(x) 2 </pre>
<<=	<pre> >>> x=0b1100 >>> x<<=1 >>> print(bin(x)) 0b11000 >>> print(x) 24 </pre>

Command line arguments

The values send from command prompt to python program are called command line arguments.

These command line arguments send from Operating System.

The values which send from command prompt are of type string.

Python virtual machine (PVM) store all these values into one predefined variable called “argv”. This argv variable exists in sys module. “argv” is variable of type list.

Applications of command line arguments

1. Developing command line utility
Example: developing installers
2. Dynamic configuration of program
Example: providing networking details at runtime
Providing database details at runtime
3. Developing commands

Example:

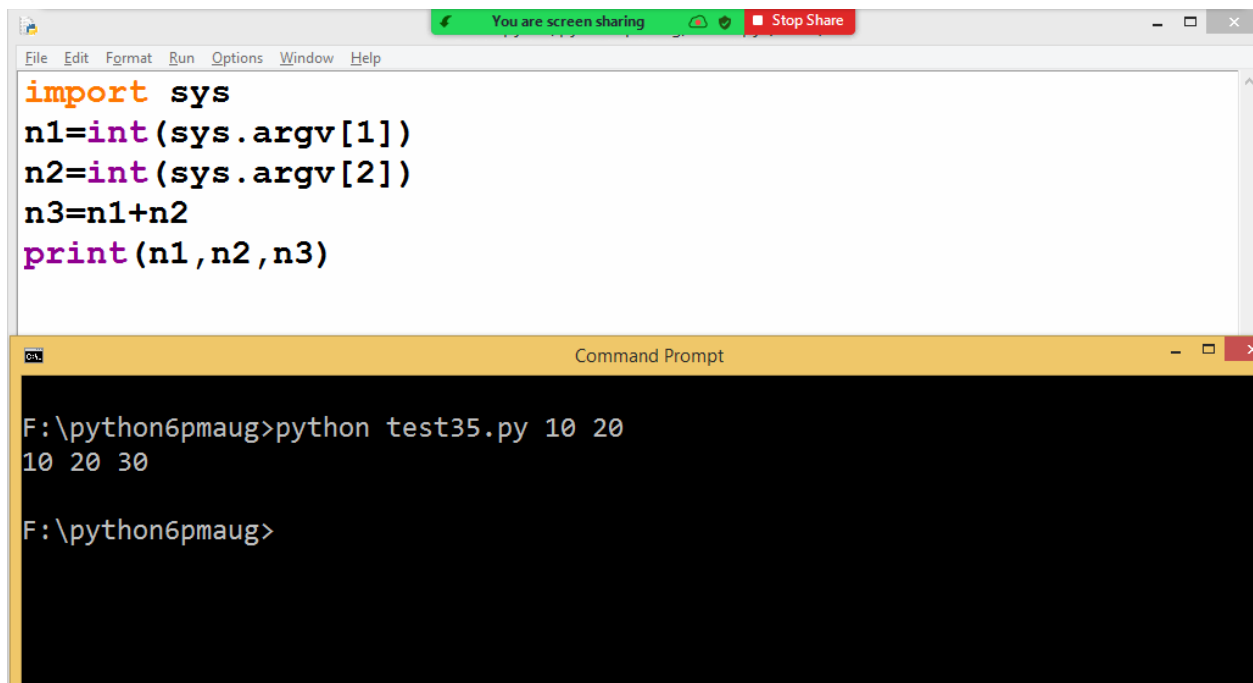
```
import sys
print(sys.argv)
```

```
F:\python6pmaug>python test34.py
['test34.py']

F:\python6pmaug>python test34.py 10 20 30 40 50
['test34.py', '10', '20', '30', '40', '50']

F:\python6pmaug>_
```

Example:



The image shows a Python script in a text editor and its execution in a Command Prompt. The script is as follows:

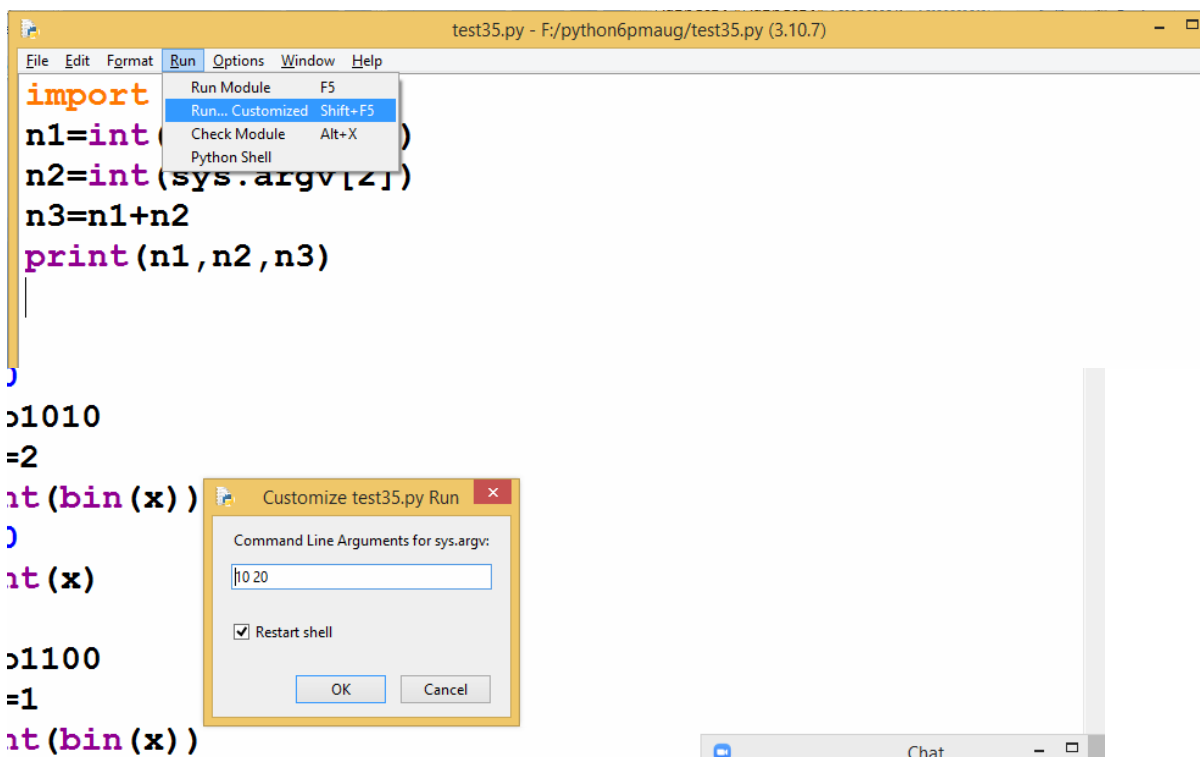
```
import sys
n1=int(sys.argv[1])
n2=int(sys.argv[2])
n3=n1+n2
print(n1,n2,n3)
```

The Command Prompt shows the execution of the script with arguments 10 and 20, resulting in the output 10 20 30.

```
F:\python6pmaug>python test35.py 10 20
10 20 30

F:\python6pmaug>
```

How to test program with command line arguments within IDLE?



Index

Control Statements

- Conditional control statements

 - o If
 - o If-else
 - o If-elif-else
 - o Nested-if

- Loop control statements

 - o for
 - o while
 - o Nested loops

- Branching statements

 - o Break
 - o Continue
 - o Pass
 - o Return

- Case studies