ECE-203 – Programming for Engineers

Contact

Dr. James Shackleford
shack@drexel.edu
Bossone 211

Office Hours: 3 – 4 pm (Tuesday)
Course Website: http://learn.dcollege.net

Textbook

Think Python
by Allen Downey
O'Reilly Press, 2015
ISBN-13: 978-1449330729
(Freely available in PDF format, check course website)

Grading

- 10% In-lab Programming Assignments
- 10% Take-Home Programming Assignments
- 35% Mid-term Exam
- 45% Final Exam

Course Philosophy

GOAL 1

Actually learn Python… for real.

If you know how the language you are programming in actually works, you are unstoppable.

Course Philosophy

GOAL 1

Actually learn Python… for real.

If you know how the language you are programming in actually works, you are unstoppable.

GOAL 2

Solve numerical problems ...algorithmically

Focus on simulation, numerical methods, and heuristic methods of problem solving.

Course Philosophy

GOAL 1

Actually learn Python… for real.

If you know how the language you are programming in actually works, you are unstoppable.

GOAL 2

Solve numerical problems ...algorithmically

Focus on simulation, numerical methods, and heuristic methods of problem solving.

GOAL 3

Utilize good coding practices

Redundant code is bad (DRY Principle)

Object-Oriented Programming (OOP)

Course Philosophy

GOAL 1

Actually learn Python… for real.

If you know how the language you are programming in actually works, you are unstoppable.

GOAL 2

Solve numerical problems ...algorithmically

Focus on simulation, numerical methods, and heuristic methods of problem solving.

GOAL 3

Utilize good coding practices

Redundant code is bad (DRY Principle)

Object-Oriented Programming (OOP)

GOAL 4

Learn to teach yourself

Programming is learned by reading other people's code that are better than you.

Find a project you like and try to understand it… it's language, you have to expose yourself to other speakers

…so that they call tell you you're wrong ;-)

Getting Setup

Accessing the

Thanos Development Server

thanos.ece.drexel.edu

Getting Setup

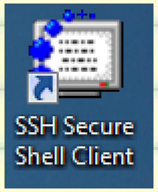
About thanos.ece.drexel.edu

- Runs Linux
- You will access Thanos using a secure shell (via ssh)
- Once logged in, you will be presented with a Bash shell
- That's right… this course is going to teach you to be awesome.

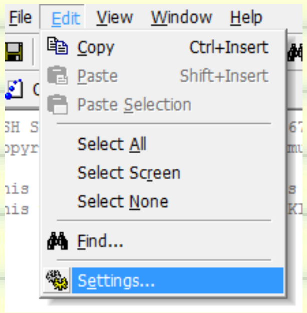
Getting Setup

Accessing from Windows (i.e. Lab Computers)

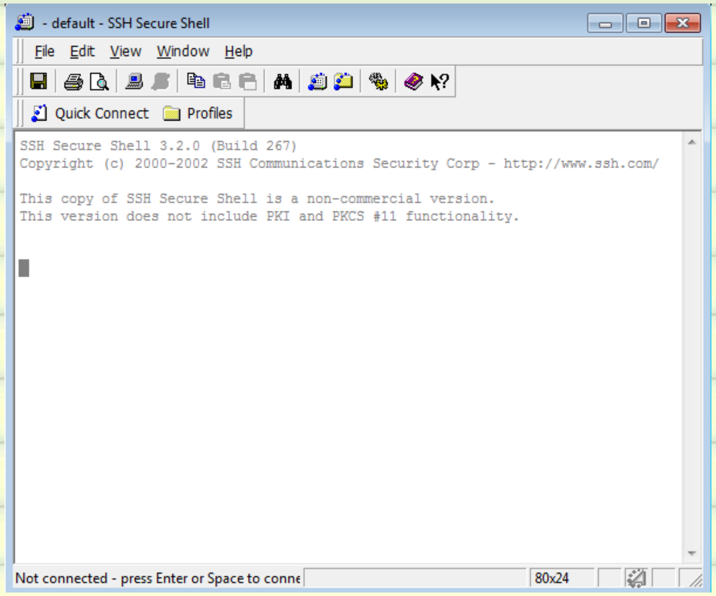
1. Launch the SSH Client



3. Go to Settings
(only have to do this once)

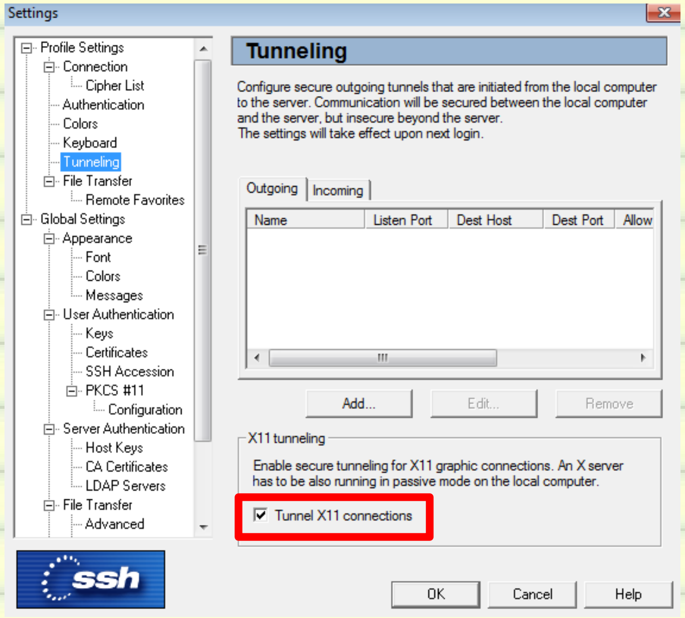


2. Stare at this beautiful window


- default - SSH Secure Shell
File
Edit
View
Window
Help
Quick Connect
Profiles
Not connected - press Enter or Space to conne
80x24

```
SSH Secure Shell 3.2.0 (Build 267)
Copyright (c) 2000-2002 SSH Communications Security Corp - http://www.ssh.com/

This copy of SSH Secure Shell is a non-commercial version.
This version does not include PKI and PKCS #11 functionality.
```

4. Make sure X11 tunneling is enabled



Tunneling

Configure secure outgoing tunnels that are initiated from the local computer to the server. Communication will be secured between the local computer and the server, but insecure beyond the server.
The settings will take effect upon next login.

X11 tunneling

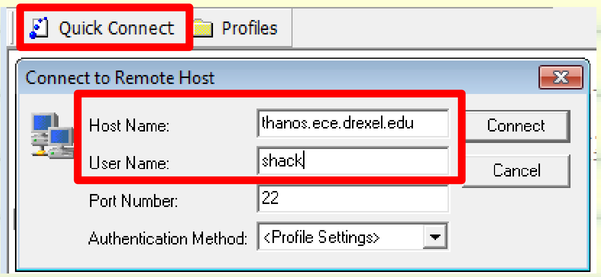
Enable secure tunneling for X11 graphic connections. An X server has to be also running in passive mode on the local computer.

[✓ Tunnel X11 connections]

Getting Setup

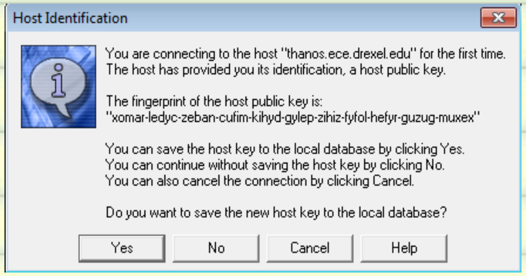
Accessing from Windows (i.e. Lab Computers)

5. Click "Quick Connect" and enter the server address & your username (Note: NOT your Drexel abc123!!)



Host Name: thanos.ece.drexel.edu
User Name: shack
Port Number: 22
Authentication Method: <Profile Settings>

6. If asked to add host public key, Yes



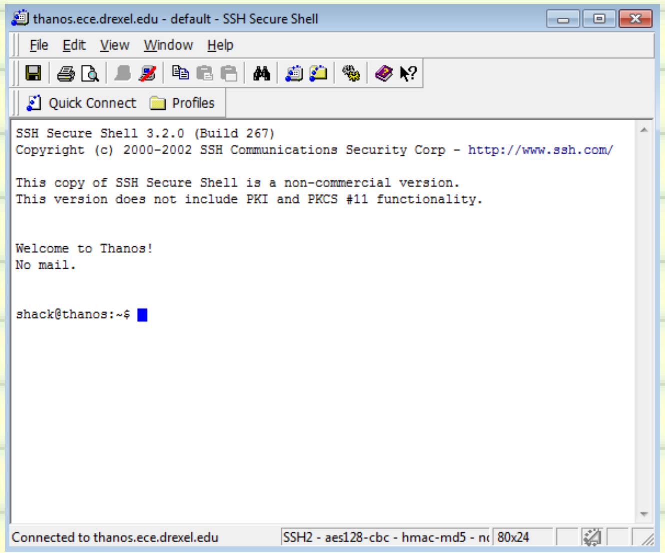
You are connecting to the host "thanos.ece.drexel.edu" for the first time. The host has provided you its identification, a host public key.

The fingerprint of the host public key is:
"xomar-ledyc-zeban-cufim-kihyd-gylep-zihiz-fyfol-hefyr-guzug-muxex"

You can save the host key to the local database by clicking Yes.
You can continue without saving the host key by clicking No.
You can also cancel the connection by clicking Cancel.

Do you want to save the new host key to the local database?

7. Ready!



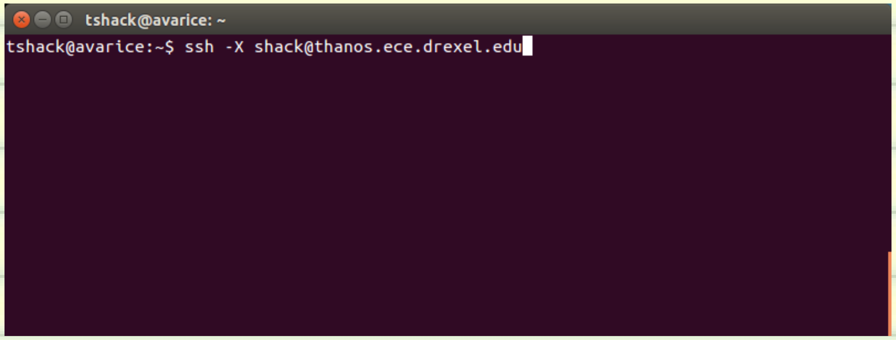
```
SSH Secure Shell 3.2.0 (Build 267)
Copyright (c) 2000-2002 SSH Communications Security Corp - http://www.ssh.com/

This copy of SSH Secure Shell is a non-commercial version.
This version does not include PKI and PKCS #11 functionality.


Welcome to Thanos!
No mail.


shack@thanos:~$
```

Connected to thanos.ece.drexel.edu | SSH2 - aes128-cbc - hmac-md5 - n | 80x24

Getting Setup

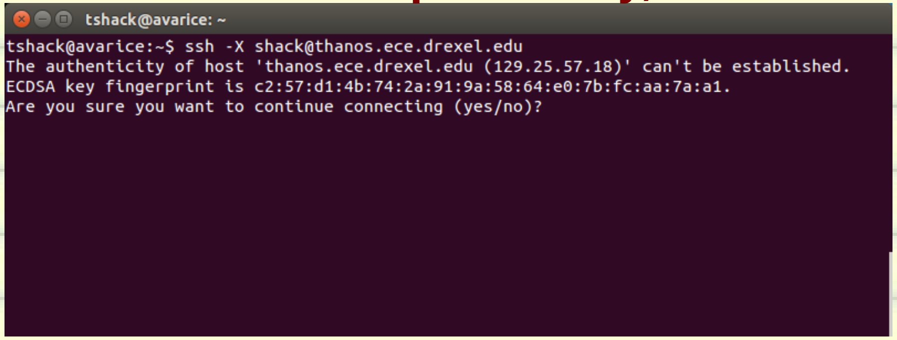
Accessing from Linux/OSX

1. Launch a Terminal & connect using ssh with your username (-X enables X11 Forwarding)



```
tshack@avarice:~$ ssh -X shack@thanos.ece.drexel.edu
```

2. If asked to add host public key, Yes

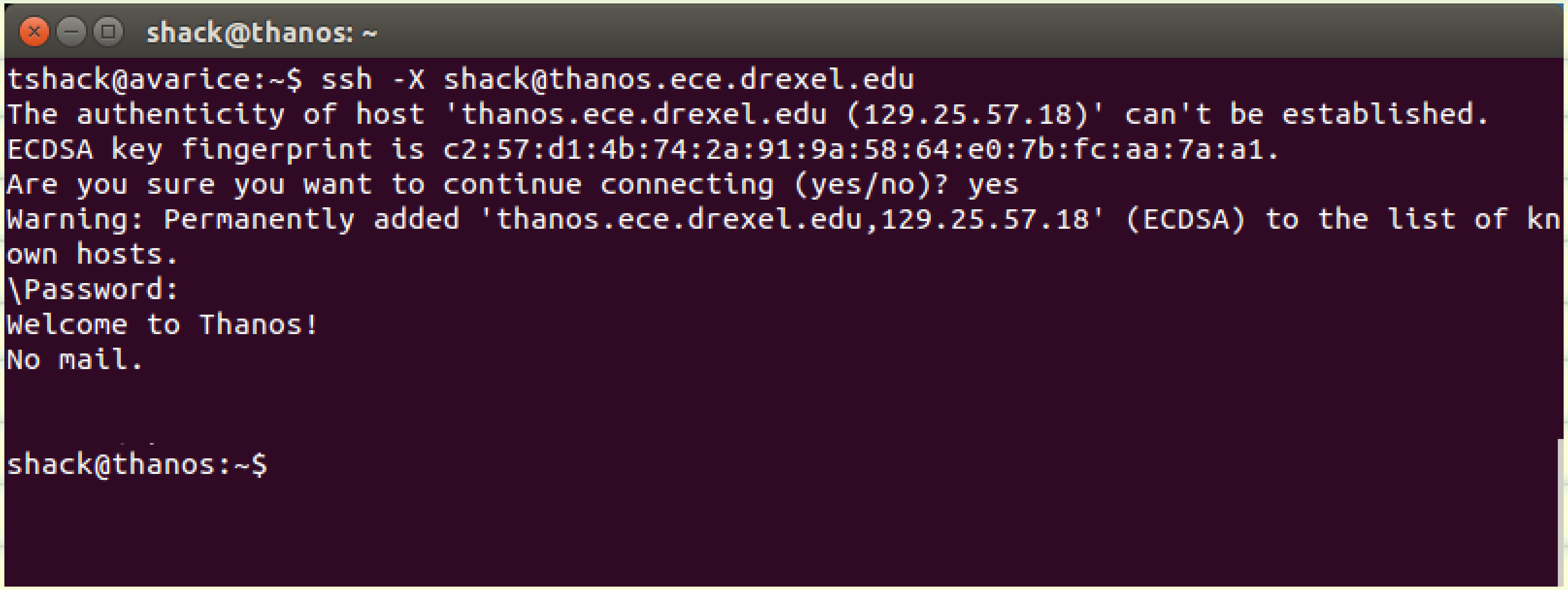


```
tshack@avarice:~$ ssh -X shack@thanos.ece.drexel.edu
The authenticity of host 'thanos.ece.drexel.edu (129.25.57.18)' can't be established.
ECDSA key fingerprint is c2:57:d1:4b:74:2a:91:9a:58:64:e0:7b:fc:aa:7a:a1.
Are you sure you want to continue connecting (yes/no)?
```

Getting Setup

Accessing from Linux/OSX

3. Ready



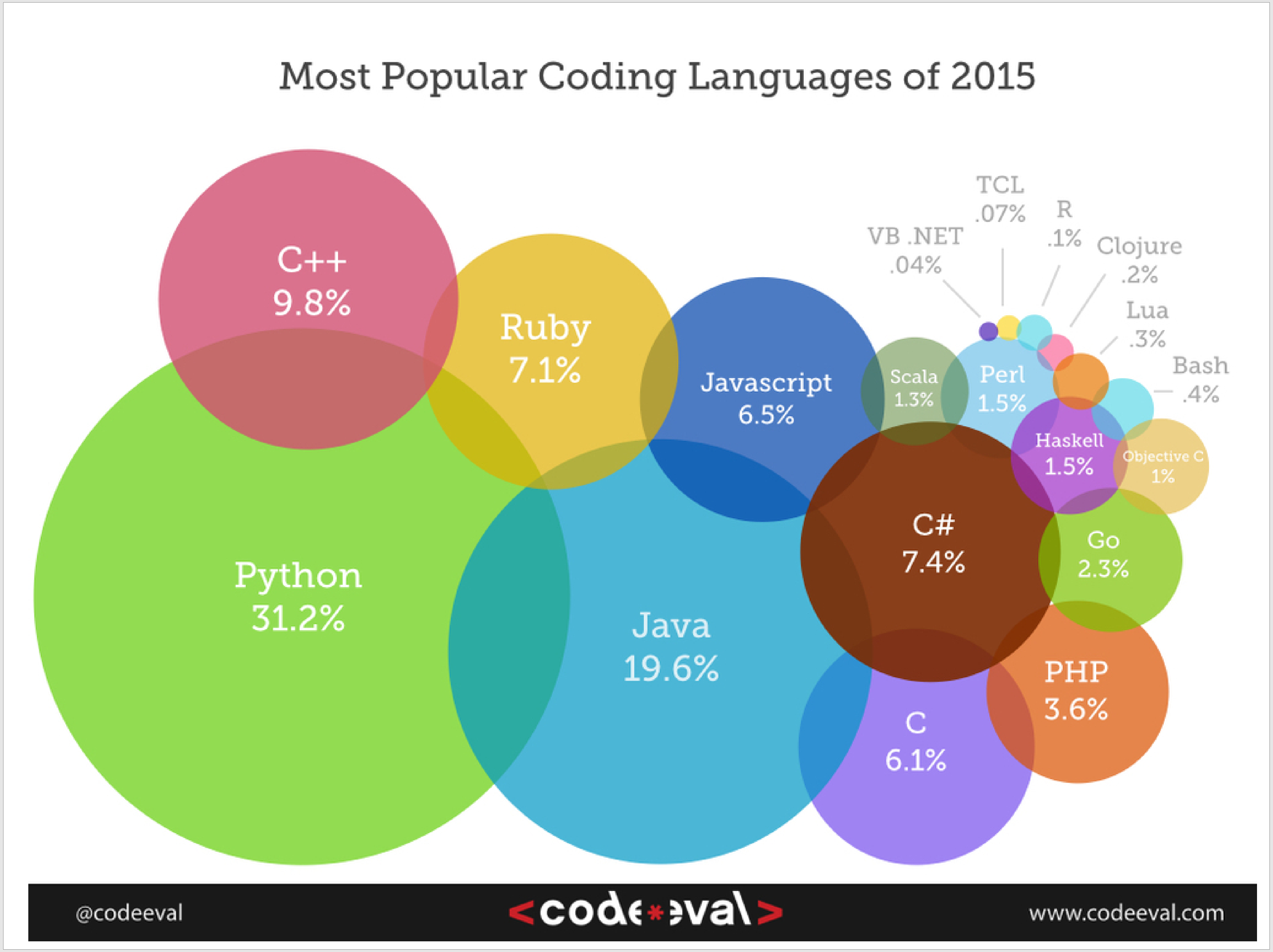
shack@thanos: ~

```
tshack@avarice:~$ ssh -X shack@thanos.ece.drexel.edu
The authenticity of host 'thanos.ece.drexel.edu (129.25.57.18)' can't be established.
ECDSA key fingerprint is c2:57:d1:4b:74:2a:91:9a:58:64:e0:7b:fc:aa:7a:a1.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'thanos.ece.drexel.edu,129.25.57.18' (ECDSA) to the list of kn
own hosts.
\Password:
Welcome to Thanos!
No mail.


shack@thanos:~$
```

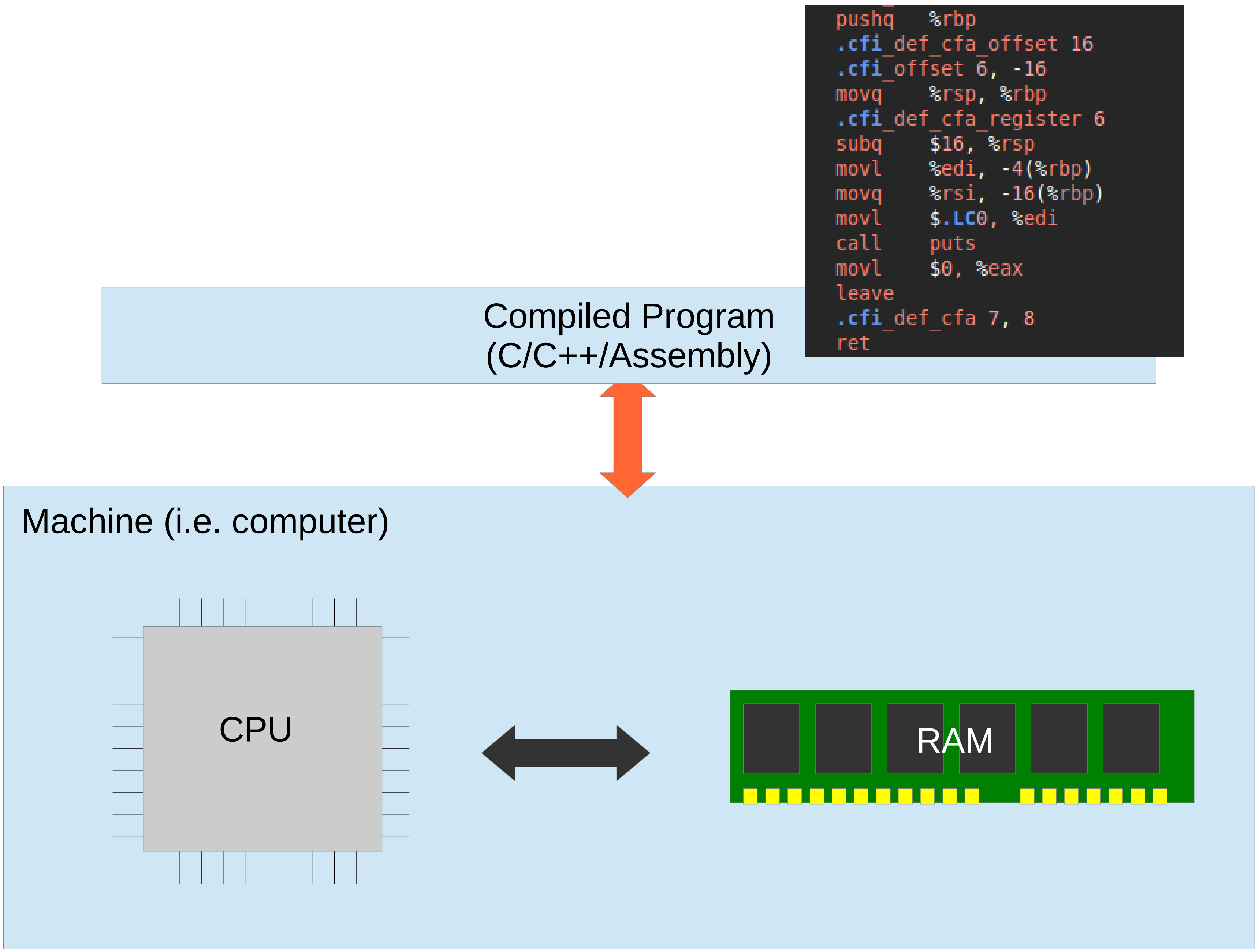
Introduction

What is Python?

Introduction

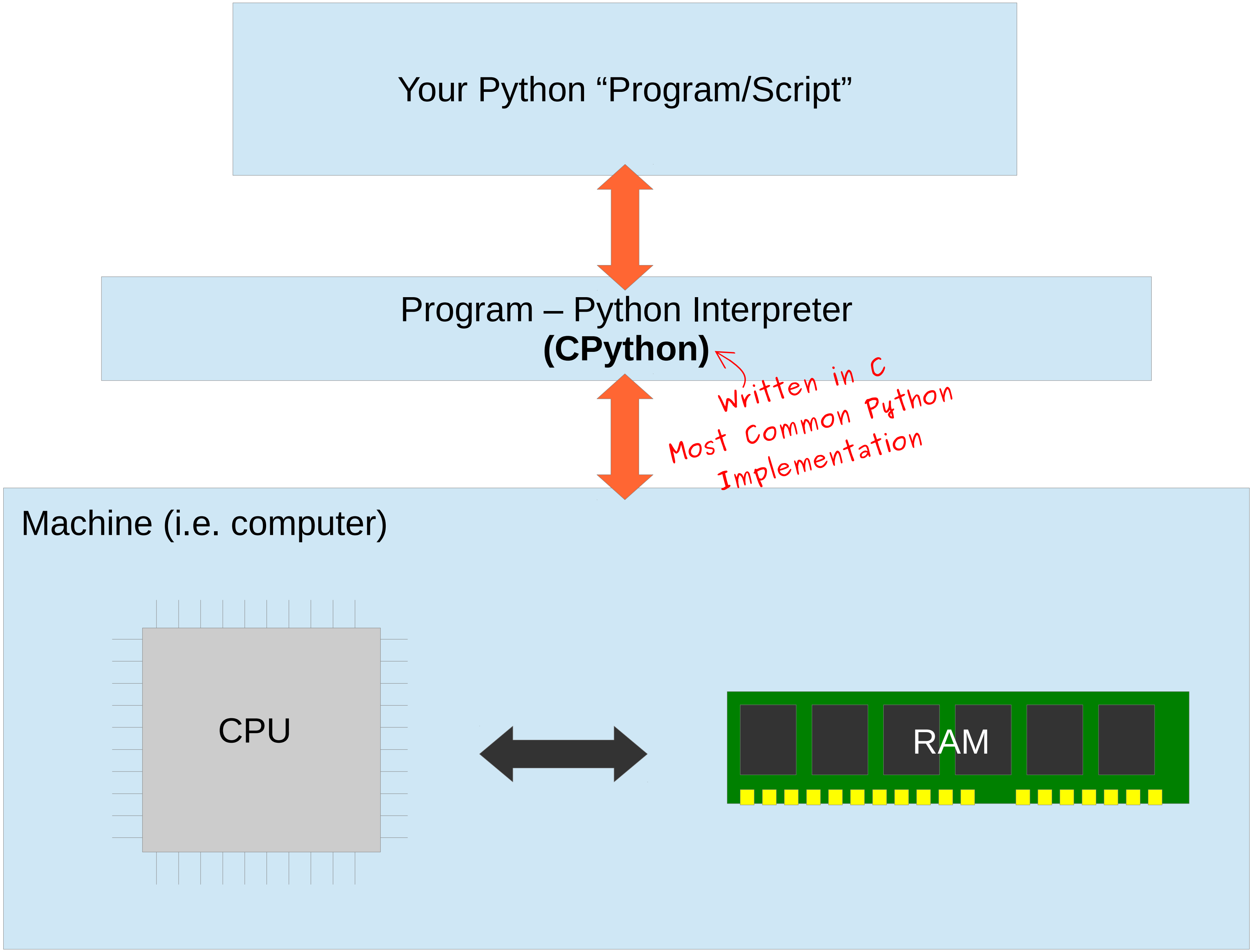


Introduction



```
pushq   %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq    %rsp, %rbp
.cfi_def_cfa_register 6
subq    $16, %rsp
movl    %edi, -4(%rbp)
movq    %rsi, -16(%rbp)
movl    $.LC0, %edi
call    puts
movl    $0, %eax
leave
.cfi_def_cfa 7, 8
ret
```

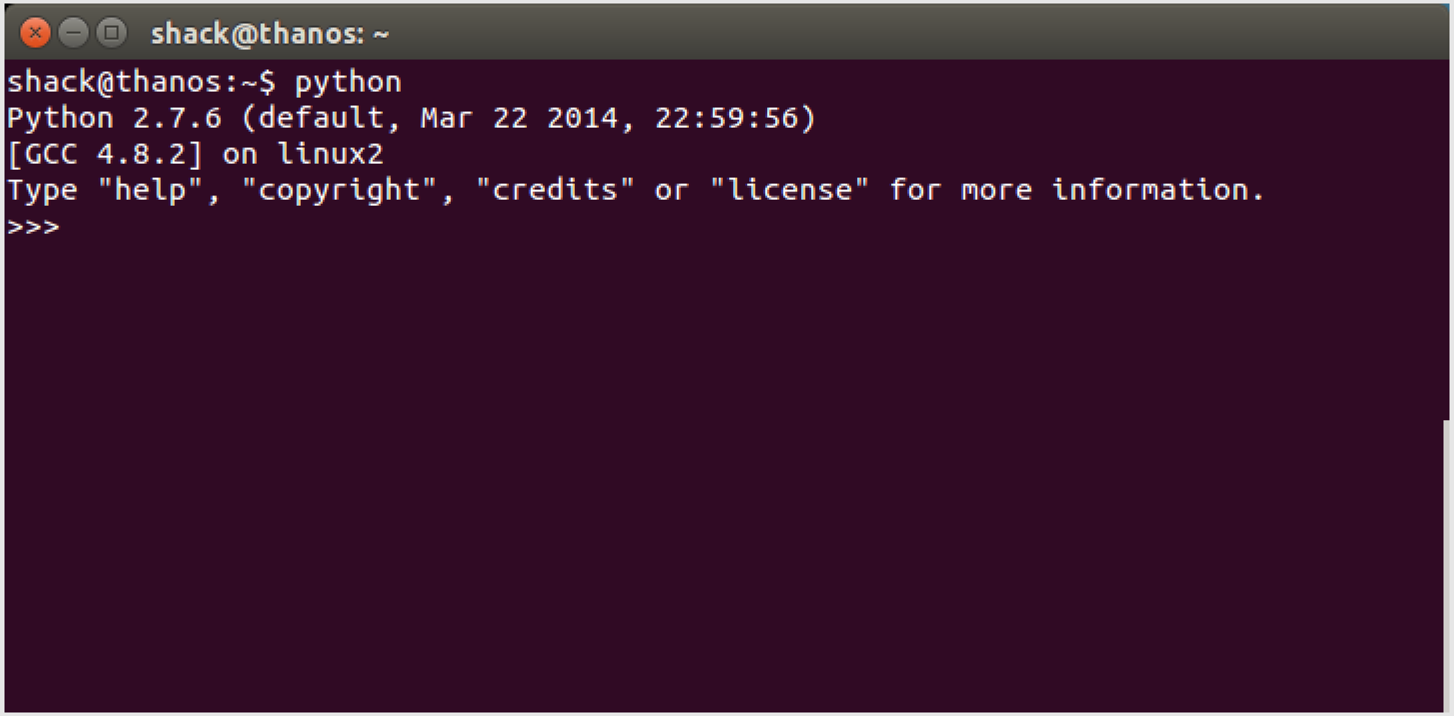
Introduction


Your Python "Program/Script"
Program – Python Interpreter
(CPython)
Written in C
Most Common Python Implementation
Machine (i.e. computer)
CPU
RAM

Introduction

Running Python

Introduction



shack@thanos: ~

```
shack@thanos:~$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

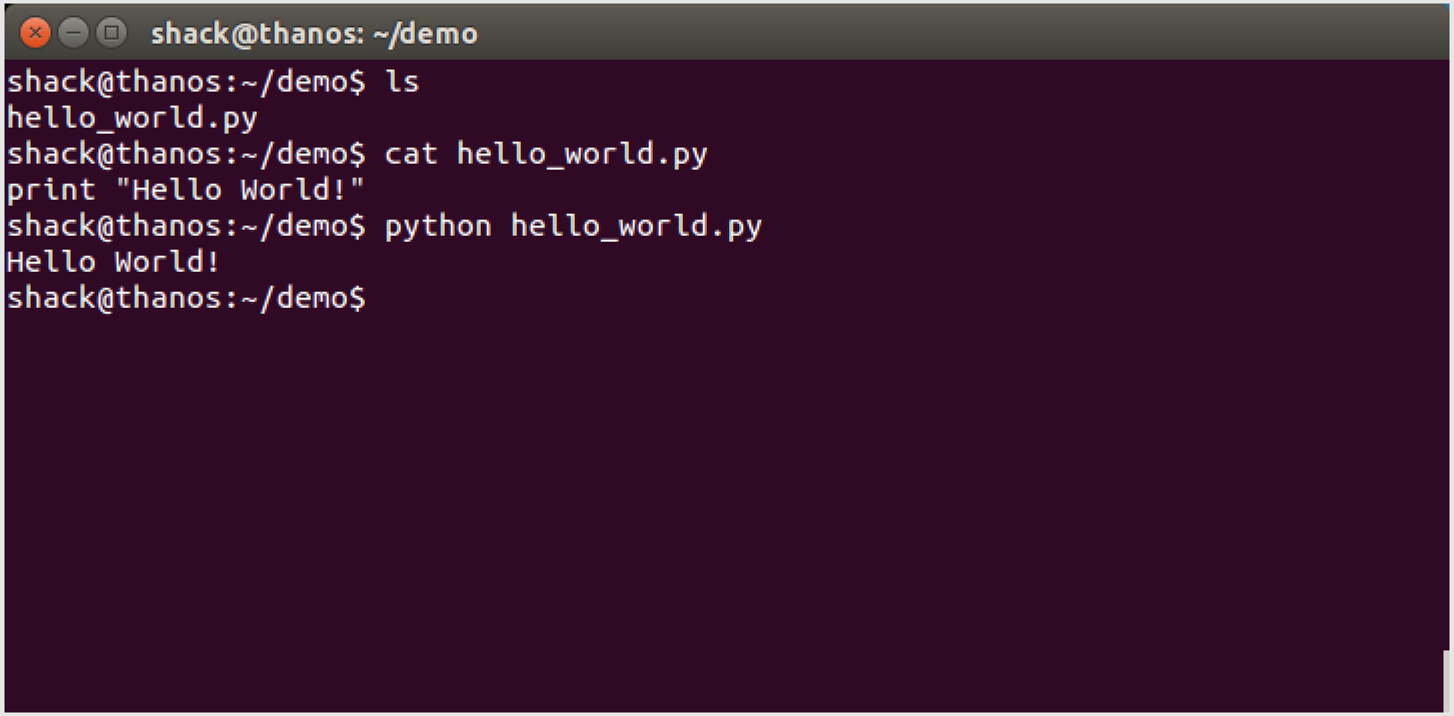
Running python without any arguments starts up an interactive command interpreter!

Good for testing code!

Bad for writing real programs.

Great for learning/testing stuff out!

Introduction



```
shack@thanos:~/demo$ ls
hello_world.py
shack@thanos:~/demo$ cat hello_world.py
print "Hello World!"
shack@thanos:~/demo$ python hello_world.py
Hello World!
shack@thanos:~/demo$
```

Supplying a python code file as an argument tells python to simply run the code in the file.

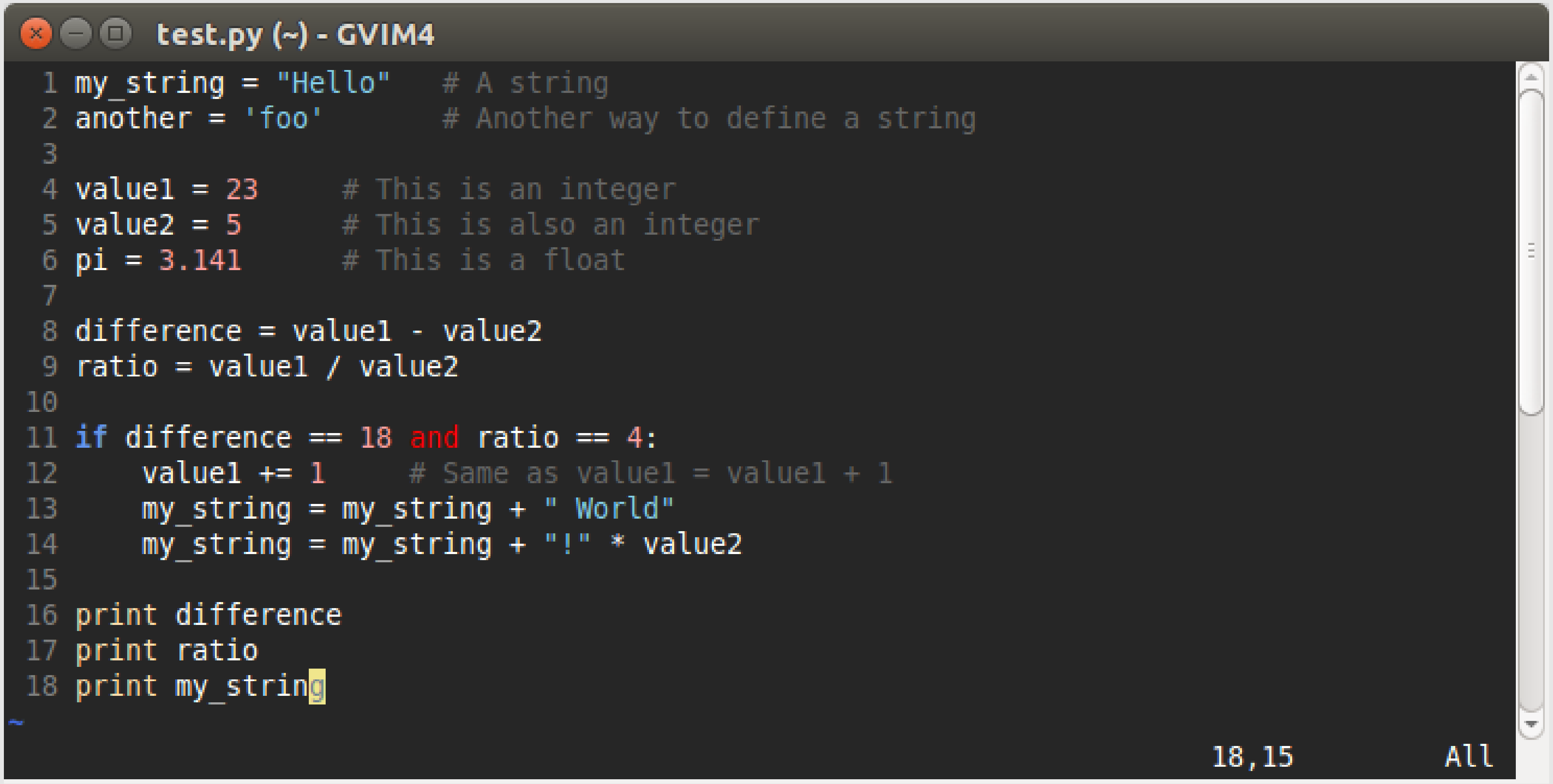
This is how “real” Python programs are executed.

Introduction

Fundamental Datatypes

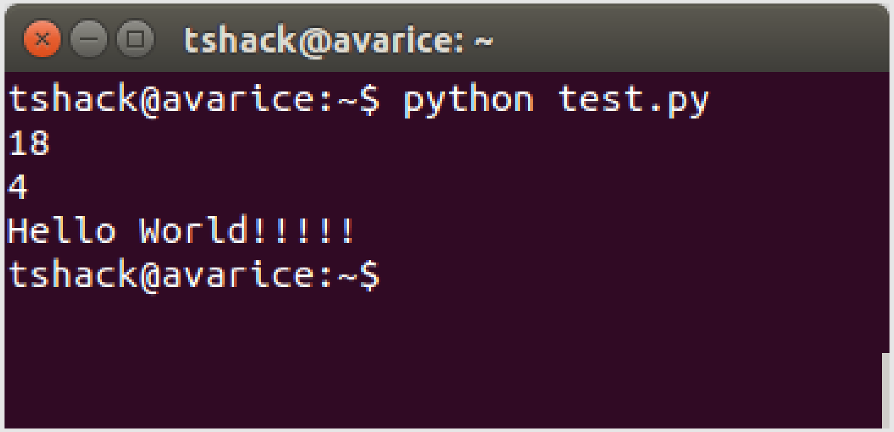
Fundamental Datatypes

First, some code to look at



```
my_string = "Hello"   # A string
another = 'foo'       # Another way to define a string

value1 = 23     # This is an integer
value2 = 5      # This is also an integer
pi = 3.141      # This is a float

difference = value1 - value2
ratio = value1 / value2

if difference == 18 and ratio == 4:
    value1 += 1     # Same as value1 = value1 + 1
    my_string = my_string + " World"
    my_string = my_string + "!" * value2

print difference
print ratio
print my_string
```

Output:

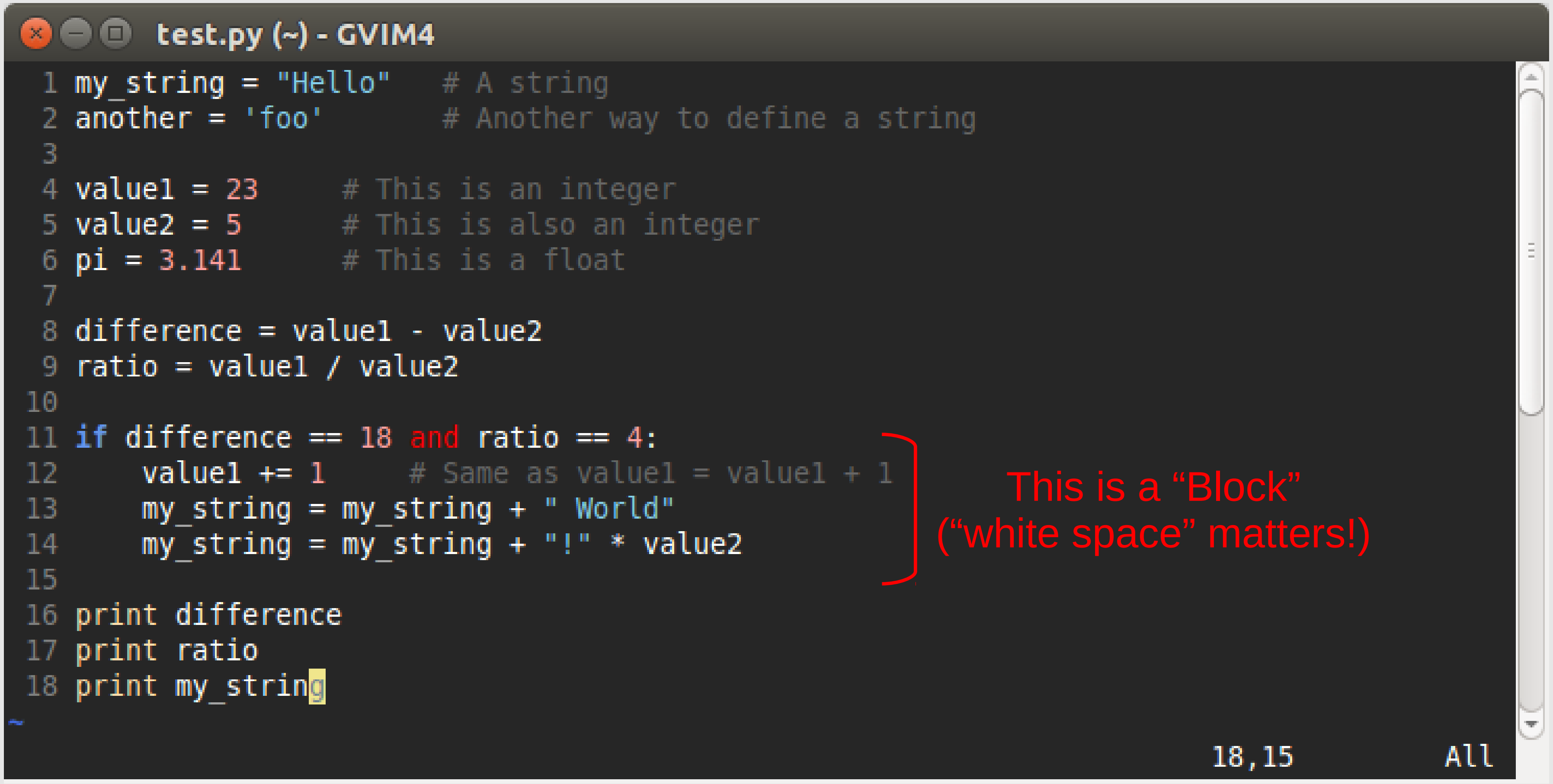


```
tshack@avarice:~$ python test.py
18
4
Hello World!!!!!
tshack@avarice:~$
```

This is a stupid program, but we can learn a lot from it!

Fundamental Datatypes

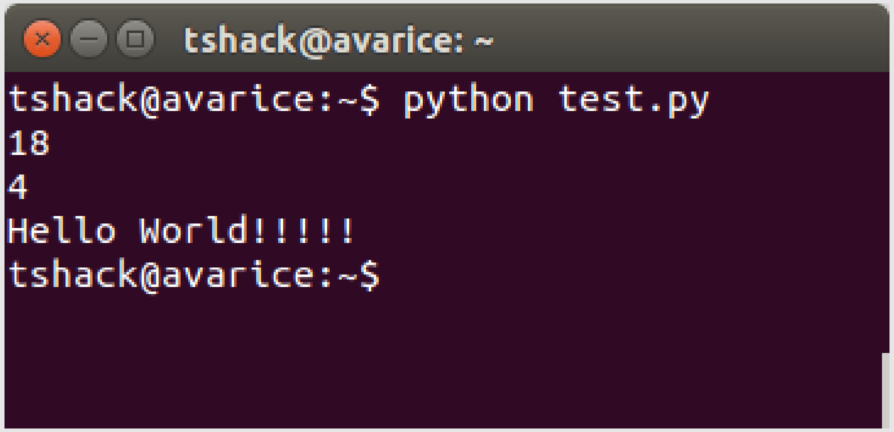
First, some code to look at



```
my_string = "Hello"   # A string
another = 'foo'       # Another way to define a string

value1 = 23      # This is an integer
value2 = 5       # This is also an integer
pi = 3.141       # This is a float

difference = value1 - value2
ratio = value1 / value2

if difference == 18 and ratio == 4:
    value1 += 1     # Same as value1 = value1 + 1
    my_string = my_string + " World"
    my_string = my_string + "!" * value2

print difference
print ratio
print my_string
```

This is a "Block"
("white space" matters!)

Output:



```
tshack@avarice:~$ python test.py
18
4
Hello World!!!!!
tshack@avarice:~$
```

This is a stupid program, but we can learn a lot from it!

The Python Execution Model

The Python Execution Model

Knowing this separates “tinkerers” from “professionals”

...but First a Question

Based ONLY on your intuition…

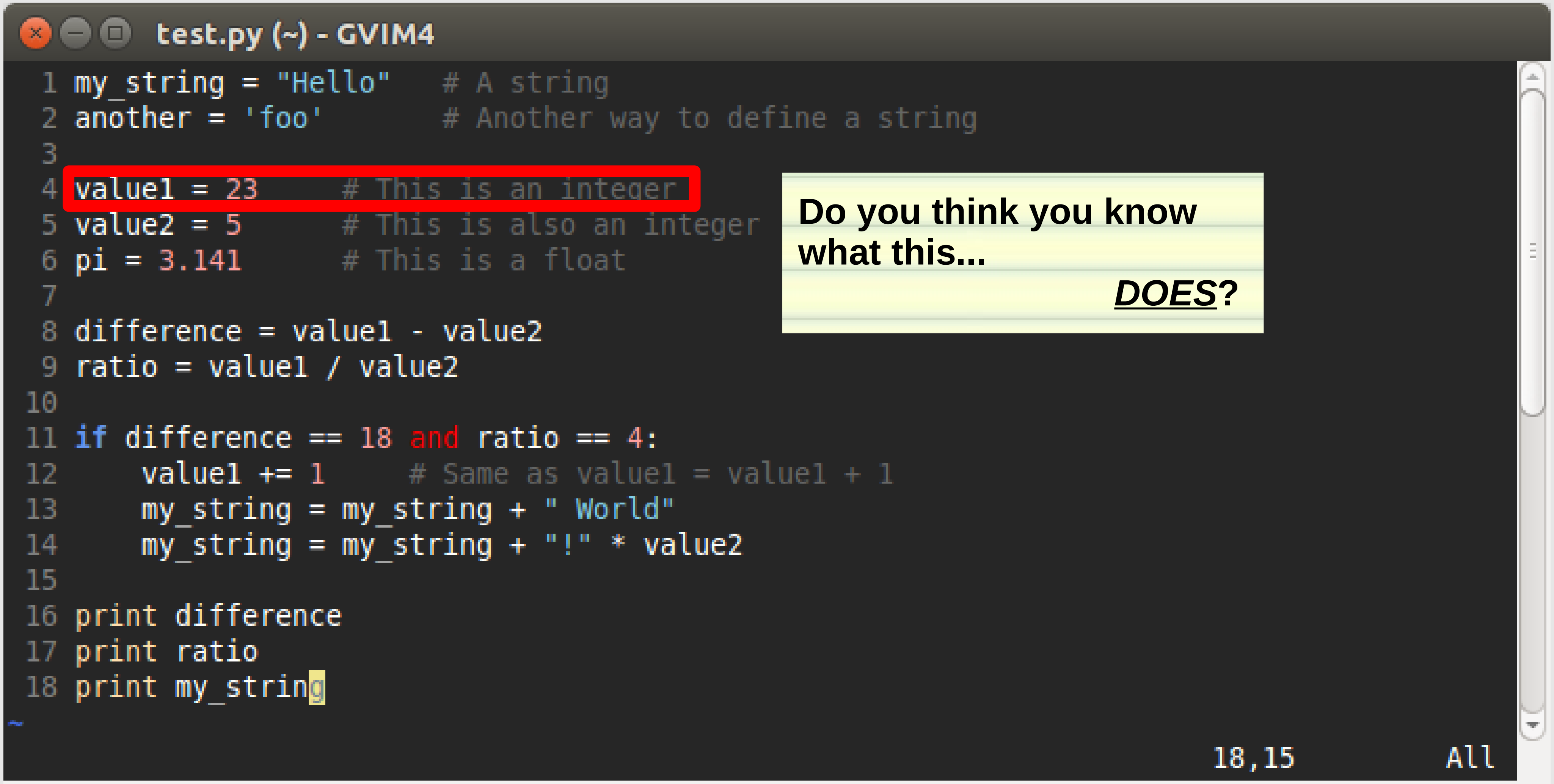
What do you think the answer is?

```
>>> foo = [23, 45, 100]
>>> bar = foo
>>> bar[1] = 1337
>>> print foo
????
```

(answer at the end of lecture)

The Python Execution Model

First, some code to look at



```
my_string = "Hello"   # A string
another = 'foo'       # Another way to define a string

value1 = 23       # This is an integer
value2 = 5        # This is also an integer
pi = 3.141        # This is a float

difference = value1 - value2
ratio = value1 / value2

if difference == 18 and ratio == 4:
    value1 += 1      # Same as value1 = value1 + 1
    my_string = my_string + " World"
    my_string = my_string + "!" * value2

print difference
print ratio
print my_string
~
                                                    18,15         All
```

Do you think you know what this... ***DOES*?**

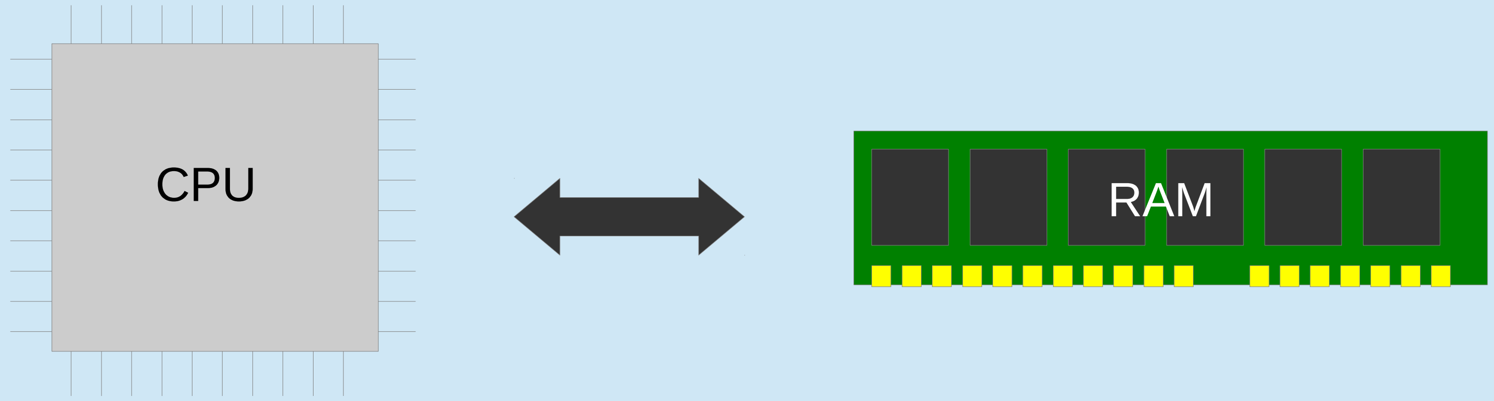
The Python Execution Model

Your Python “Program/Script”

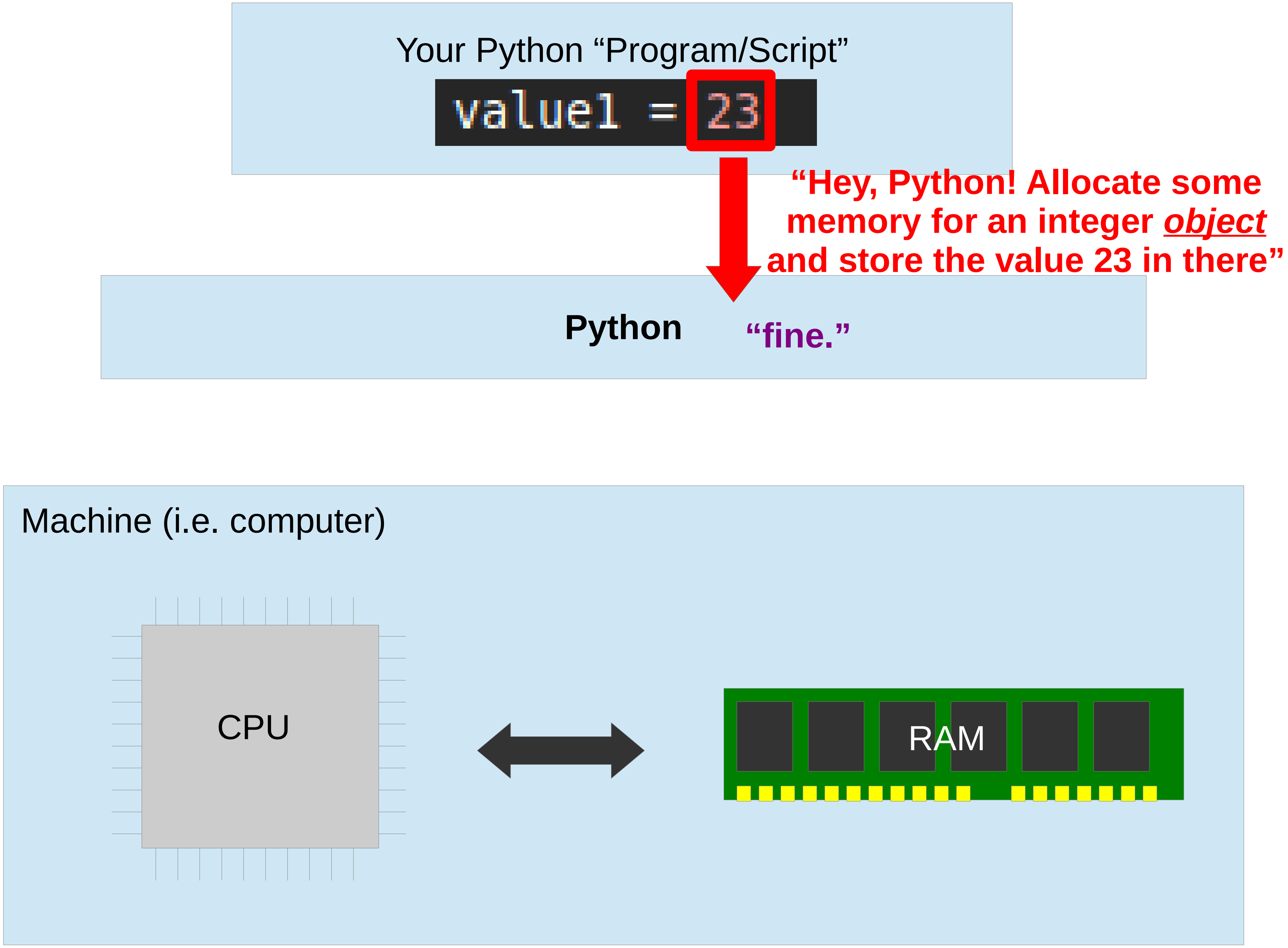
```
value1 = 23
```

Python

Machine (i.e. computer)

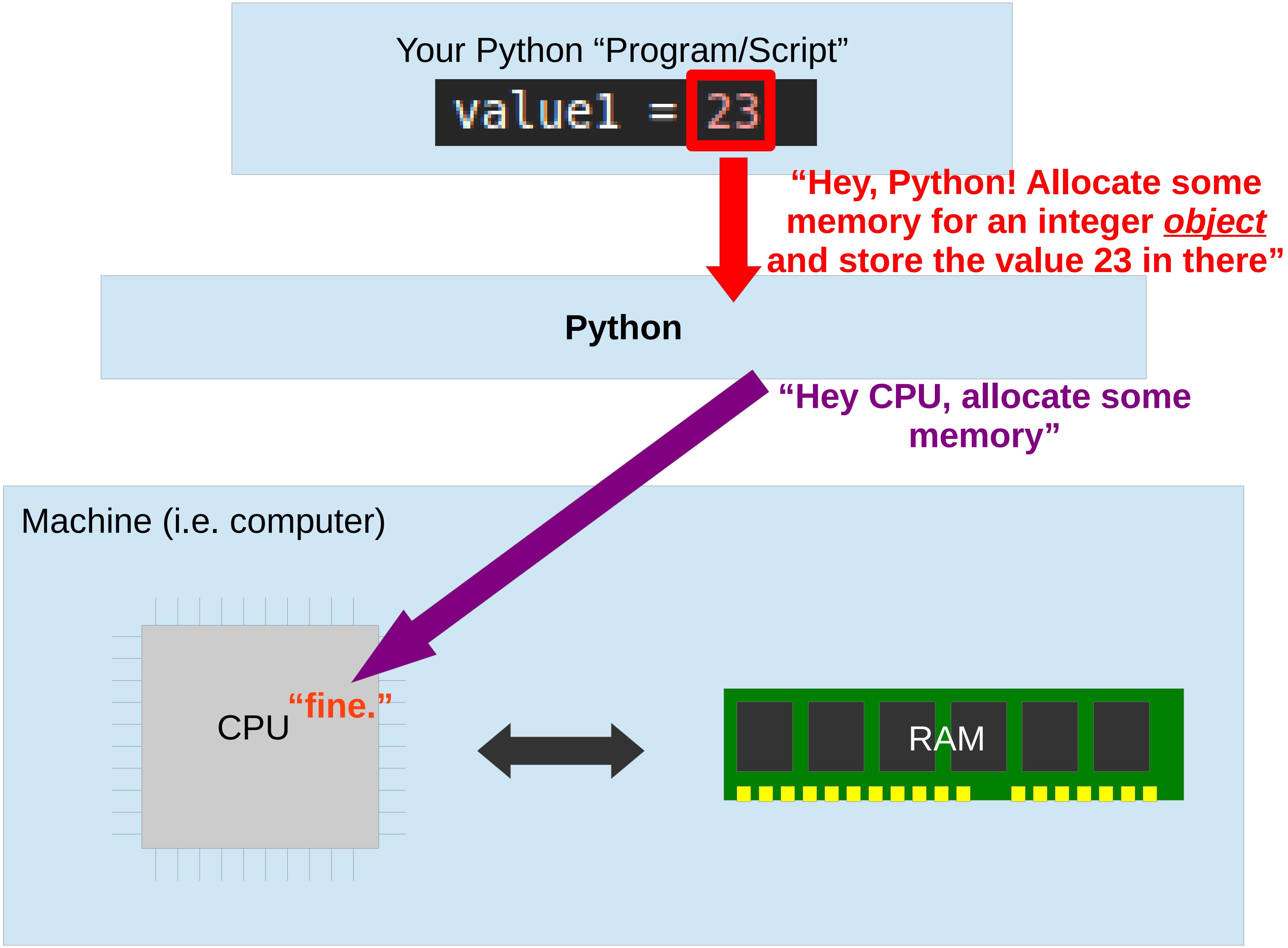


The Python Execution Model


Your Python "Program/Script"
"Hey, Python! Allocate some memory for an integer *object* and store the value 23 in there"
Python
"fine."
Machine (i.e. computer)
CPU
RAM

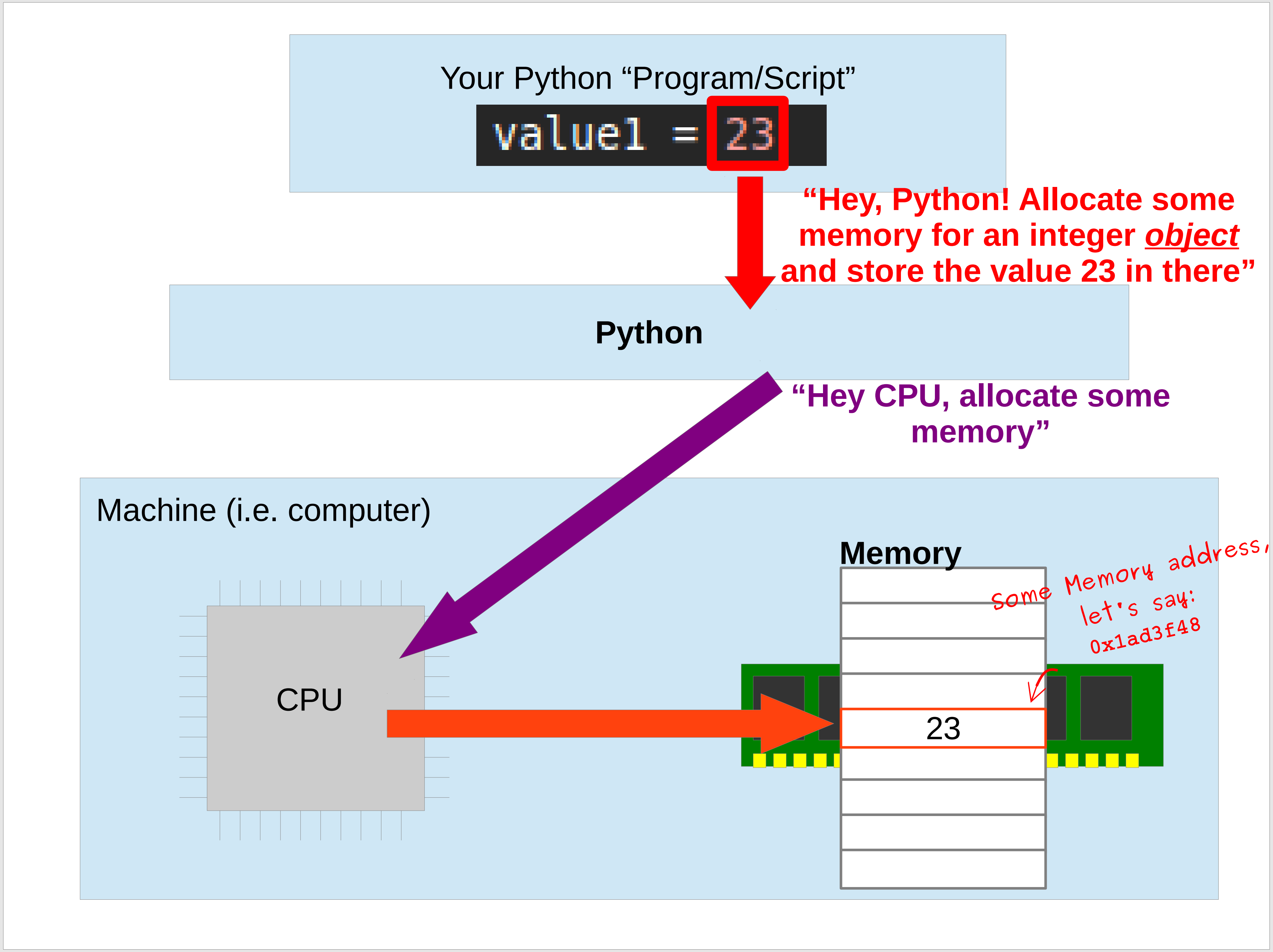
```
value1 = 23
```

The Python Execution Model

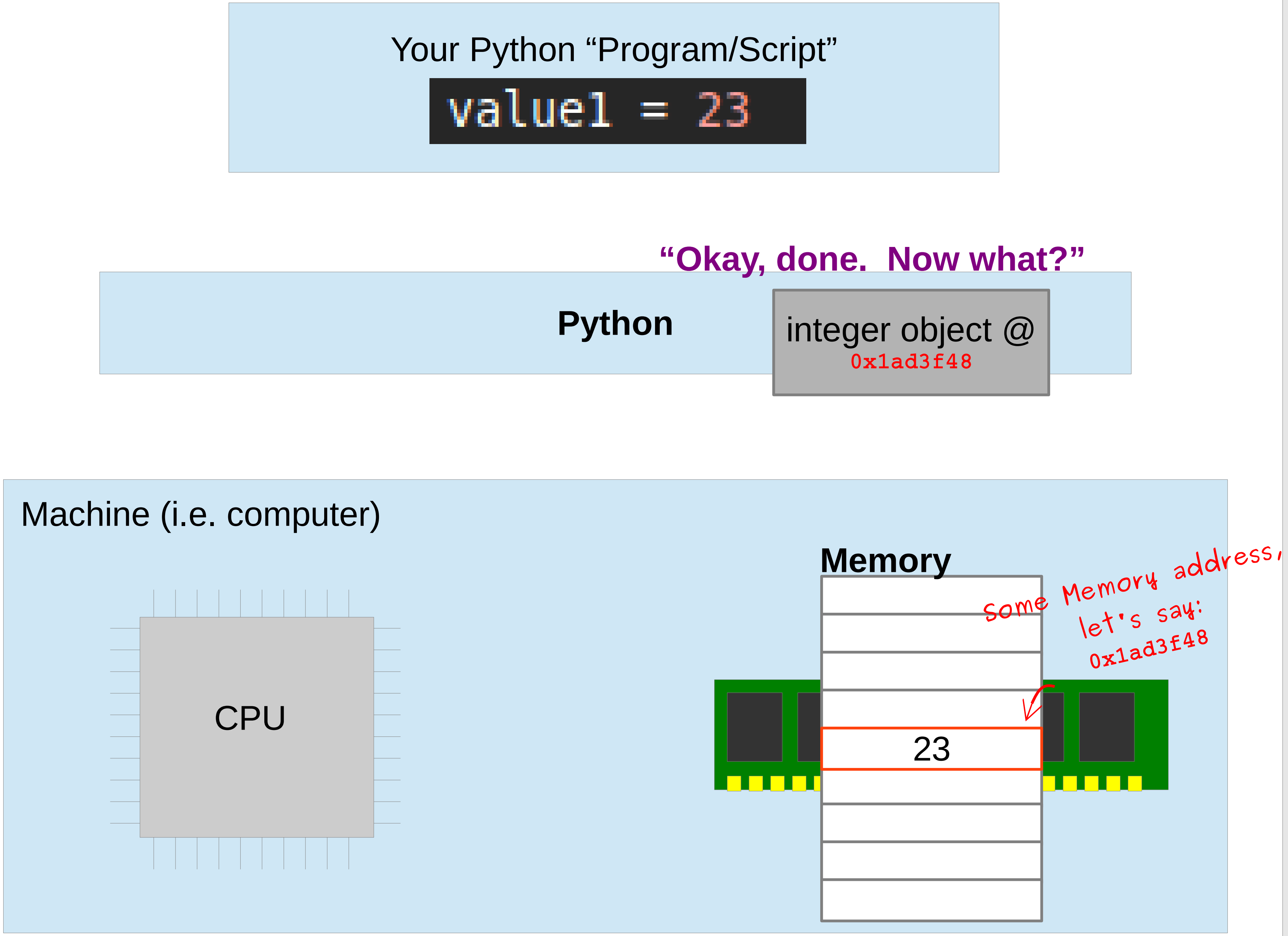

Your Python "Program/Script"
"Hey, Python! Allocate some memory for an integer *object* and store the value 23 in there"
Python
"Hey CPU, allocate some memory"
Machine (i.e. computer)
CPU
"fine."
RAM

```
value1 = 23
```

The Python Execution Model

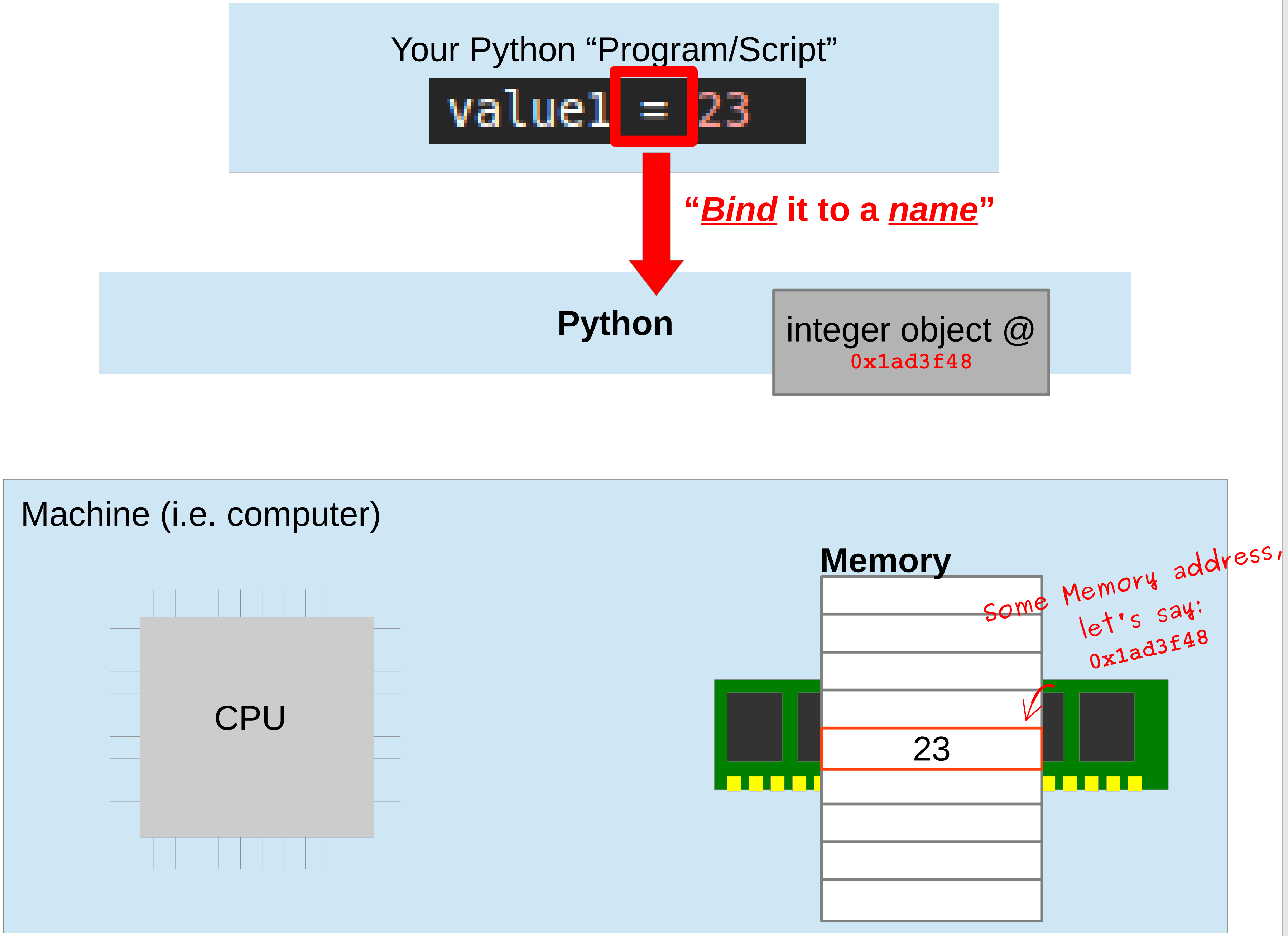

Your Python "Program/Script"
value1 = 23
"Hey, Python! Allocate some memory for an integer *object* and store the value 23 in there"
Python
"Hey CPU, allocate some memory"
Machine (i.e. computer)
Memory
CPU
23
Some Memory address, let's say: 0x1ad3f48

The Python Execution Model

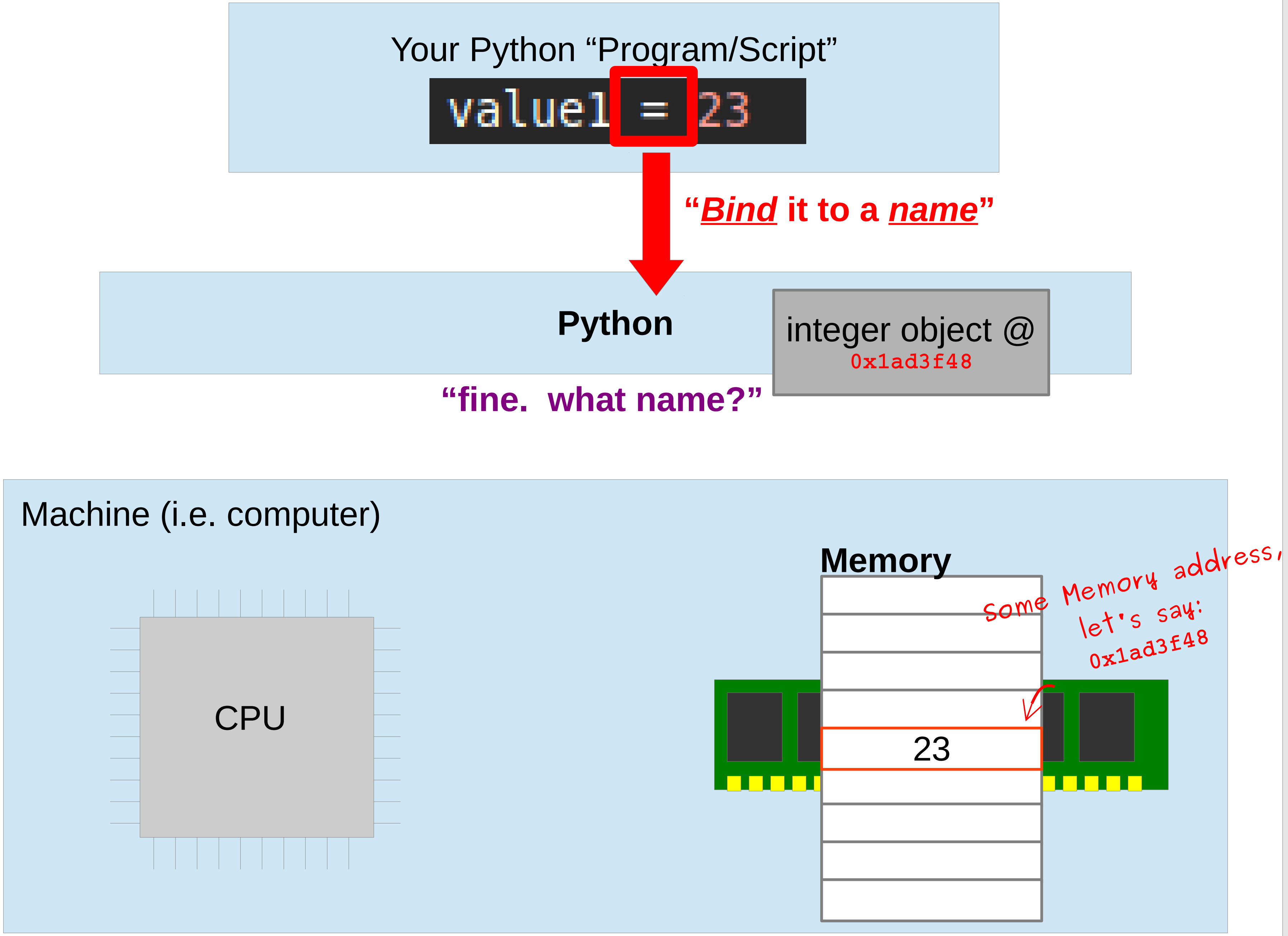


```python
value1 = 23
```

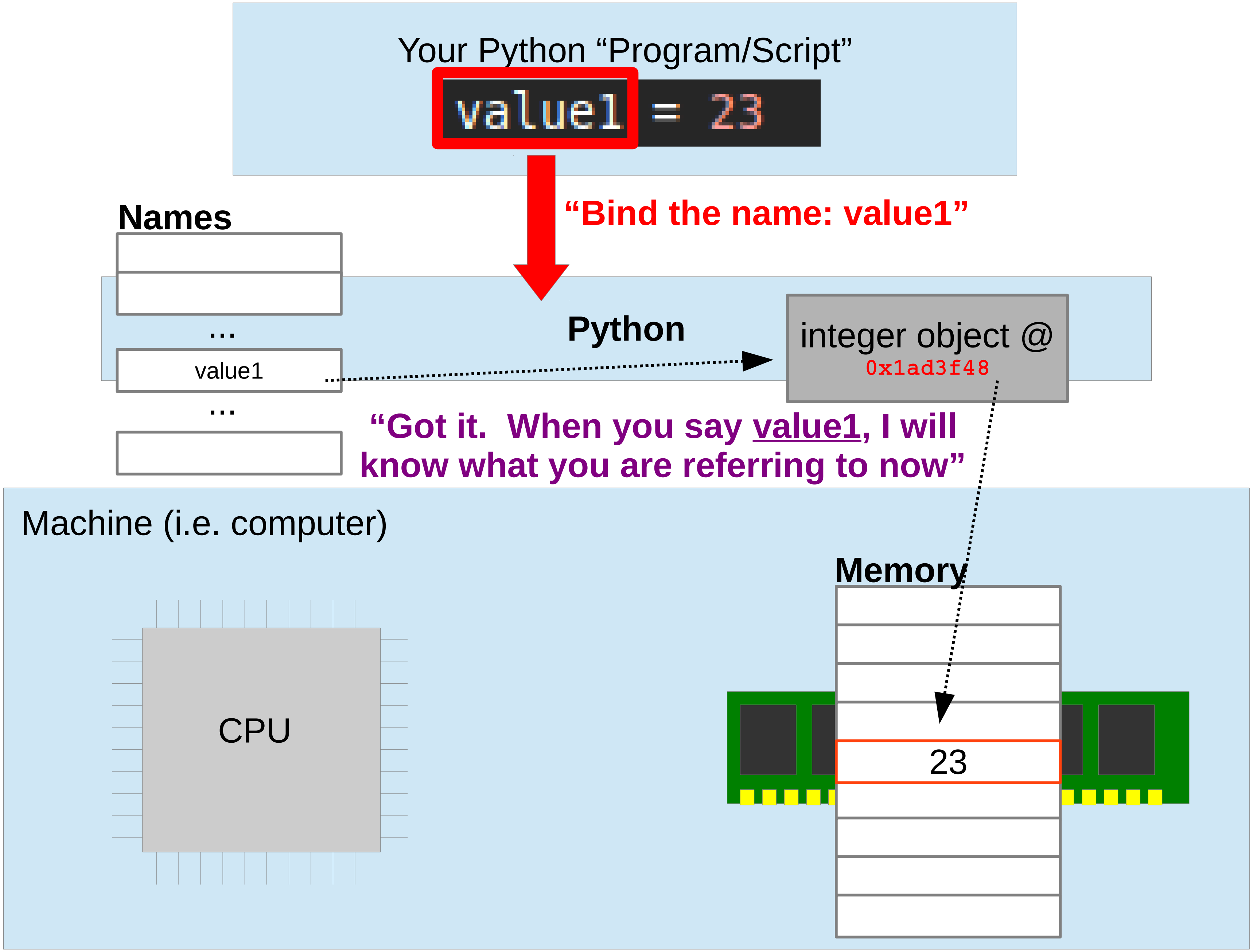
The Python Execution Model



The Python Execution Model



The Python Execution Model



Fundamental Datatypes

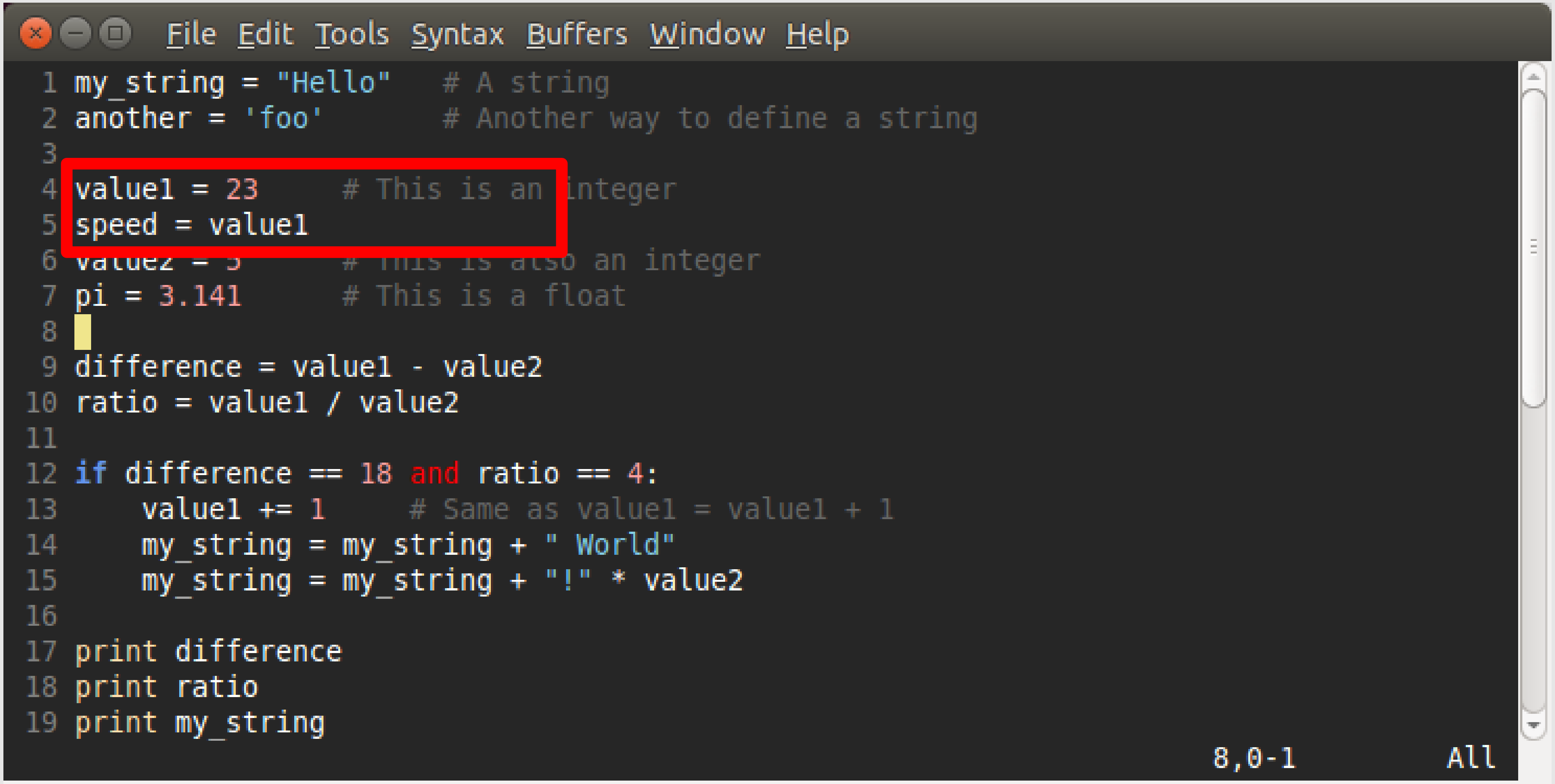
Okay, Shackleford...

Why is this important?

Because if you don't know this, the following simple code will trip you up!

Fundamental Datatypes

What happens when we encounter this?

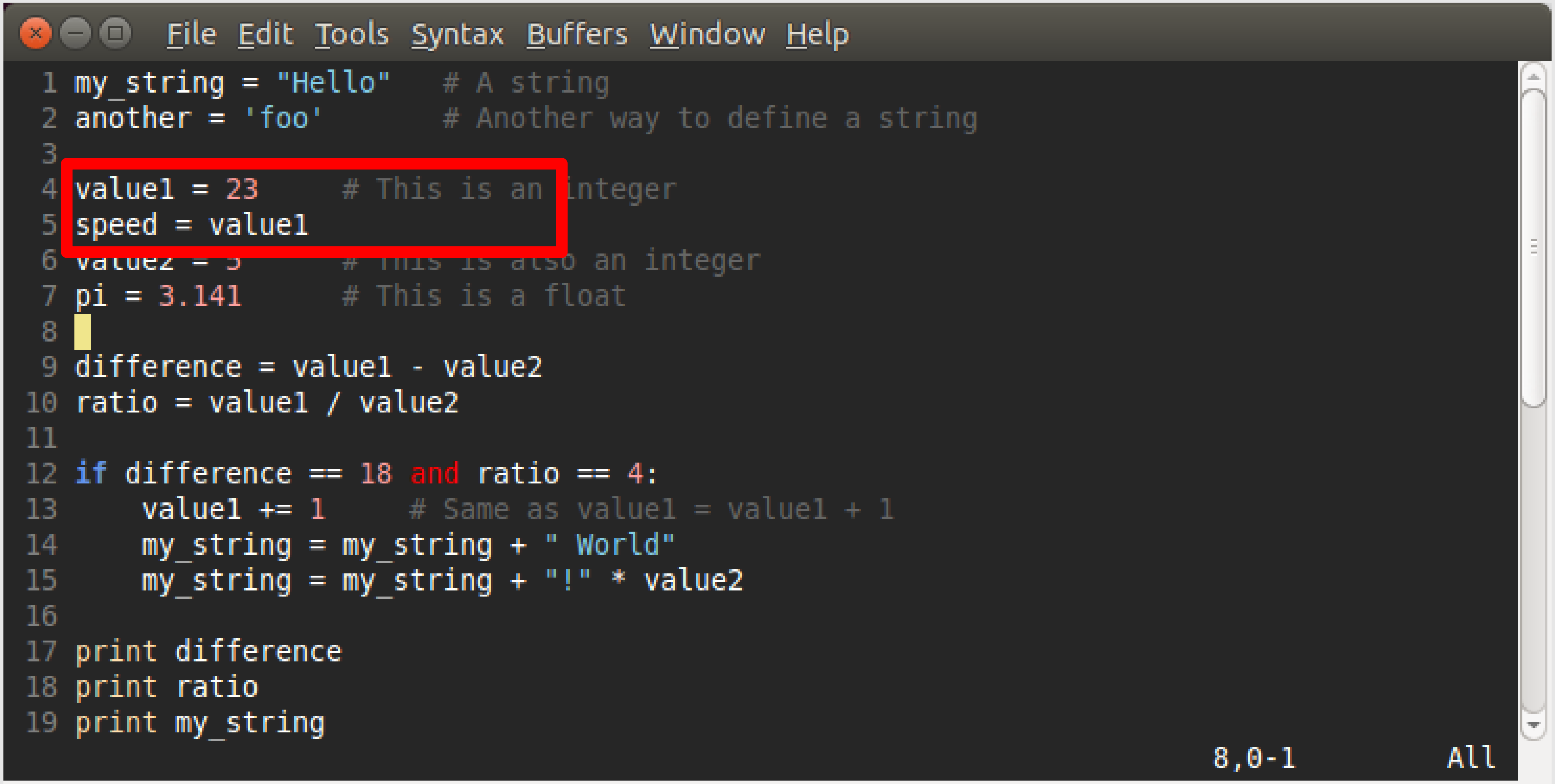


```
my_string = "Hello"   # A string
another = 'foo'       # Another way to define a string

value1 = 23     # This is an integer
speed = value1
value2 = 5      # This is also an integer
pi = 3.141      # This is a float

difference = value1 - value2
ratio = value1 / value2

if difference == 18 and ratio == 4:
    value1 += 1     # Same as value1 = value1 + 1
    my_string = my_string + " World"
    my_string = my_string + "!" * value2

print difference
print ratio
print my_string
                                                                          8,0-1         All
```

What is speed equal to?

Fundamental Datatypes

What happens when we encounter this?



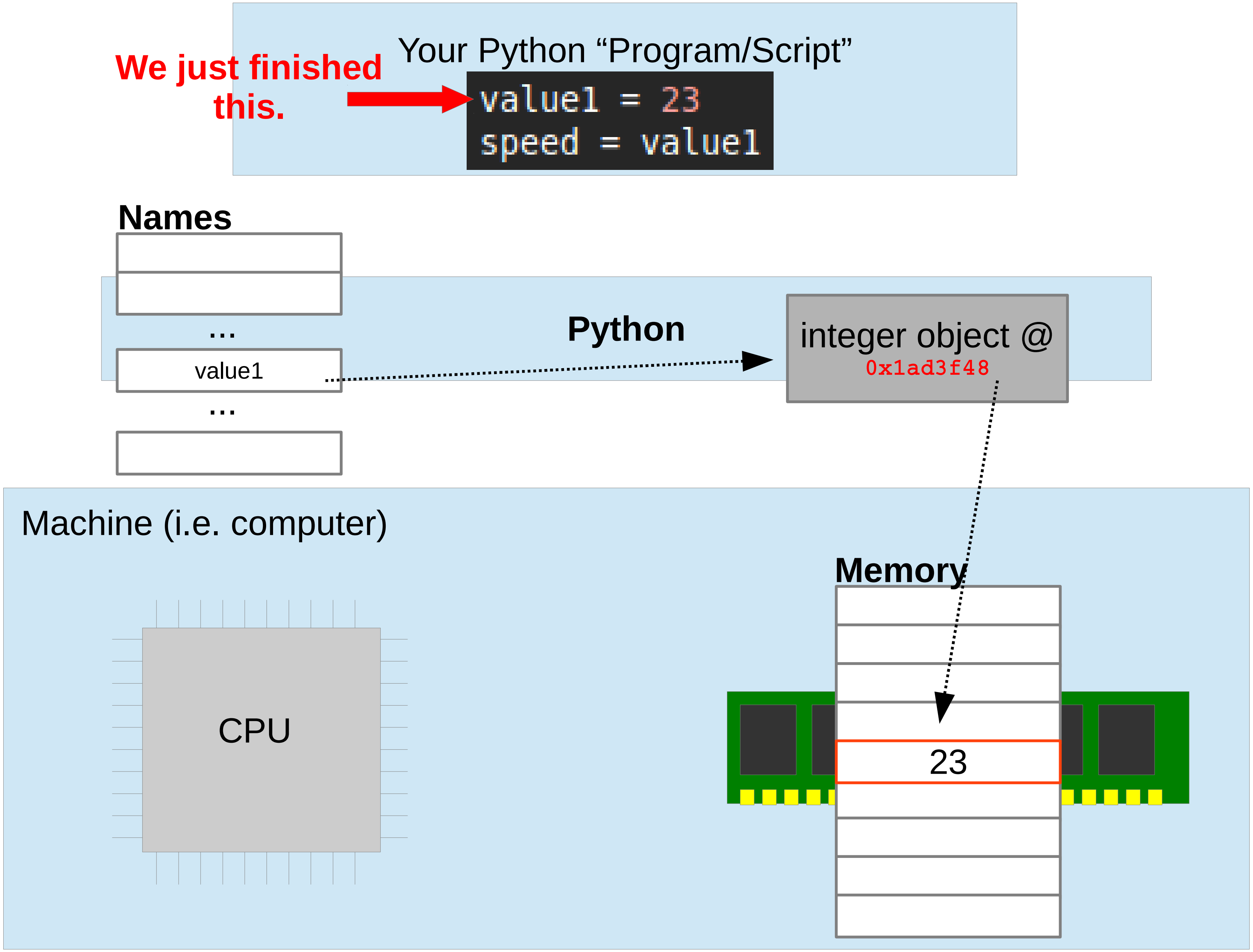
```
my_string = "Hello"   # A string
another = 'foo'       # Another way to define a string

value1 = 23      # This is an integer
speed = value1
value2 = 5       # This is also an integer
pi = 3.141       # This is a float

difference = value1 - value2
ratio = value1 / value2

if difference == 18 and ratio == 4:
    value1 += 1     # Same as value1 = value1 + 1
    my_string = my_string + " World"
    my_string = my_string + "!" * value2

print difference
print ratio
print my_string
```

What is speed equal to?

23

Fine! That *was* easy… but do you know *<u>why</u>*?

The Python Execution Model

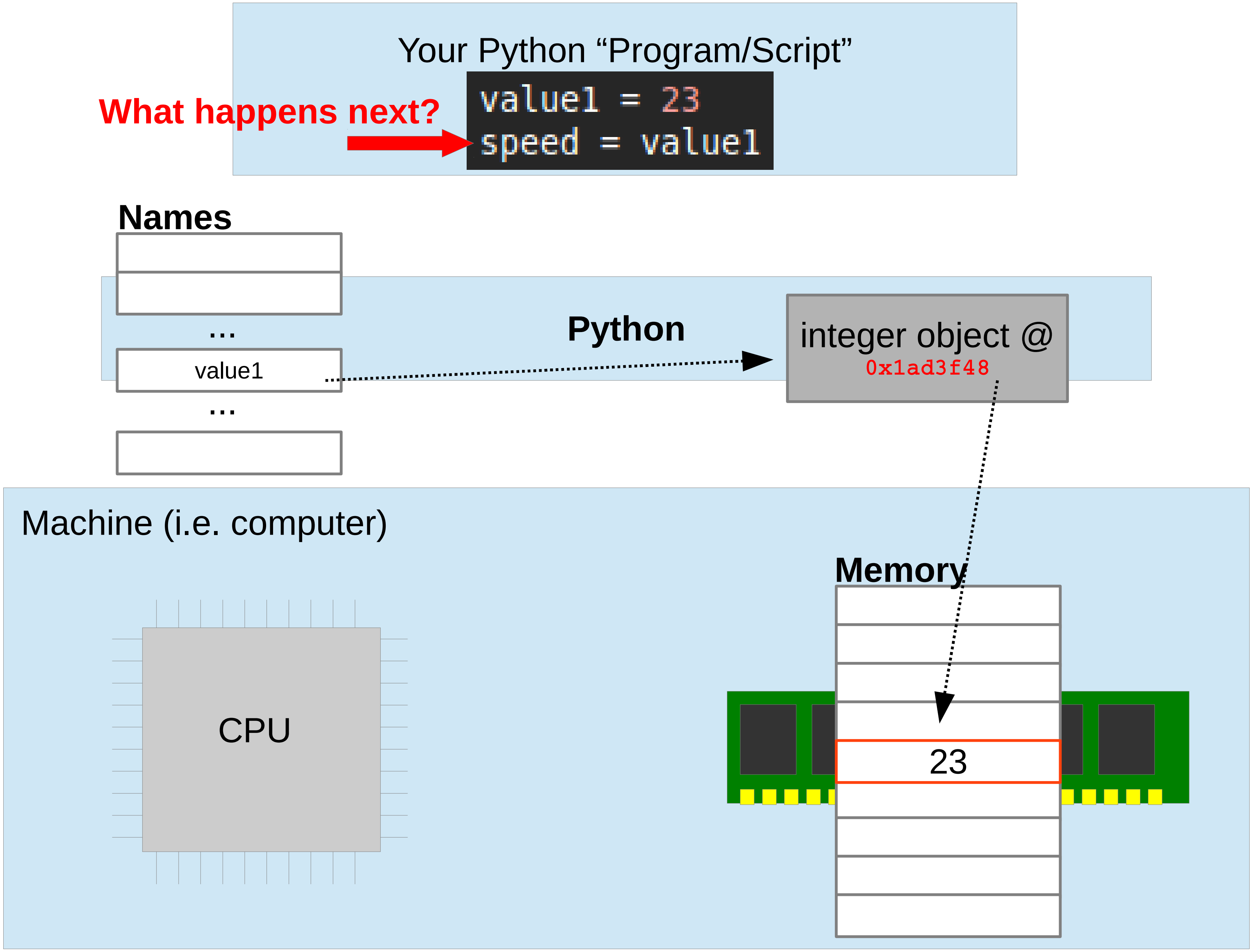


```
value1 = 23
speed = value1
```

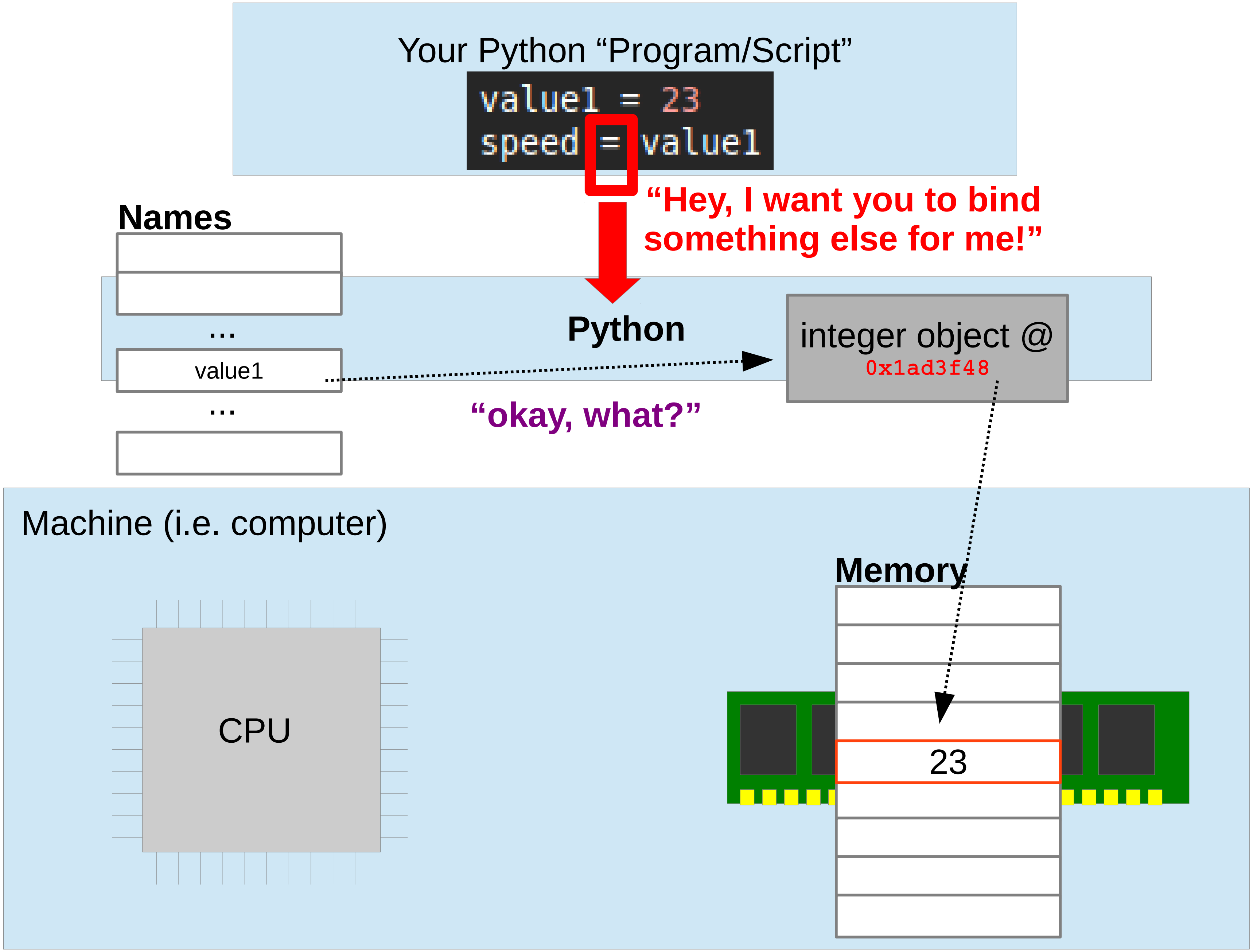
The Python Execution Model

Your Python "Program/Script"

```
value1 = 23
speed = value1
```

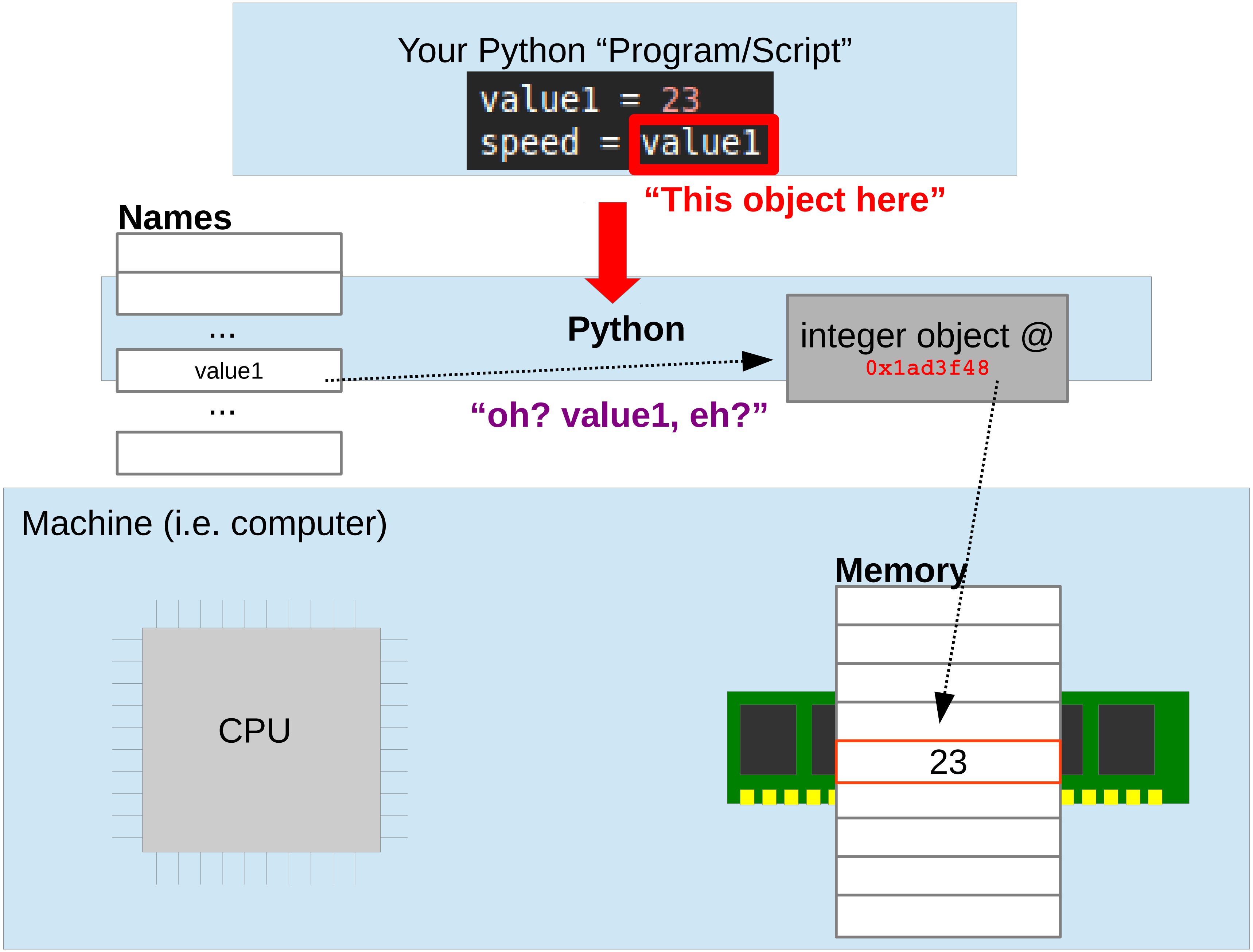


The Python Execution Model



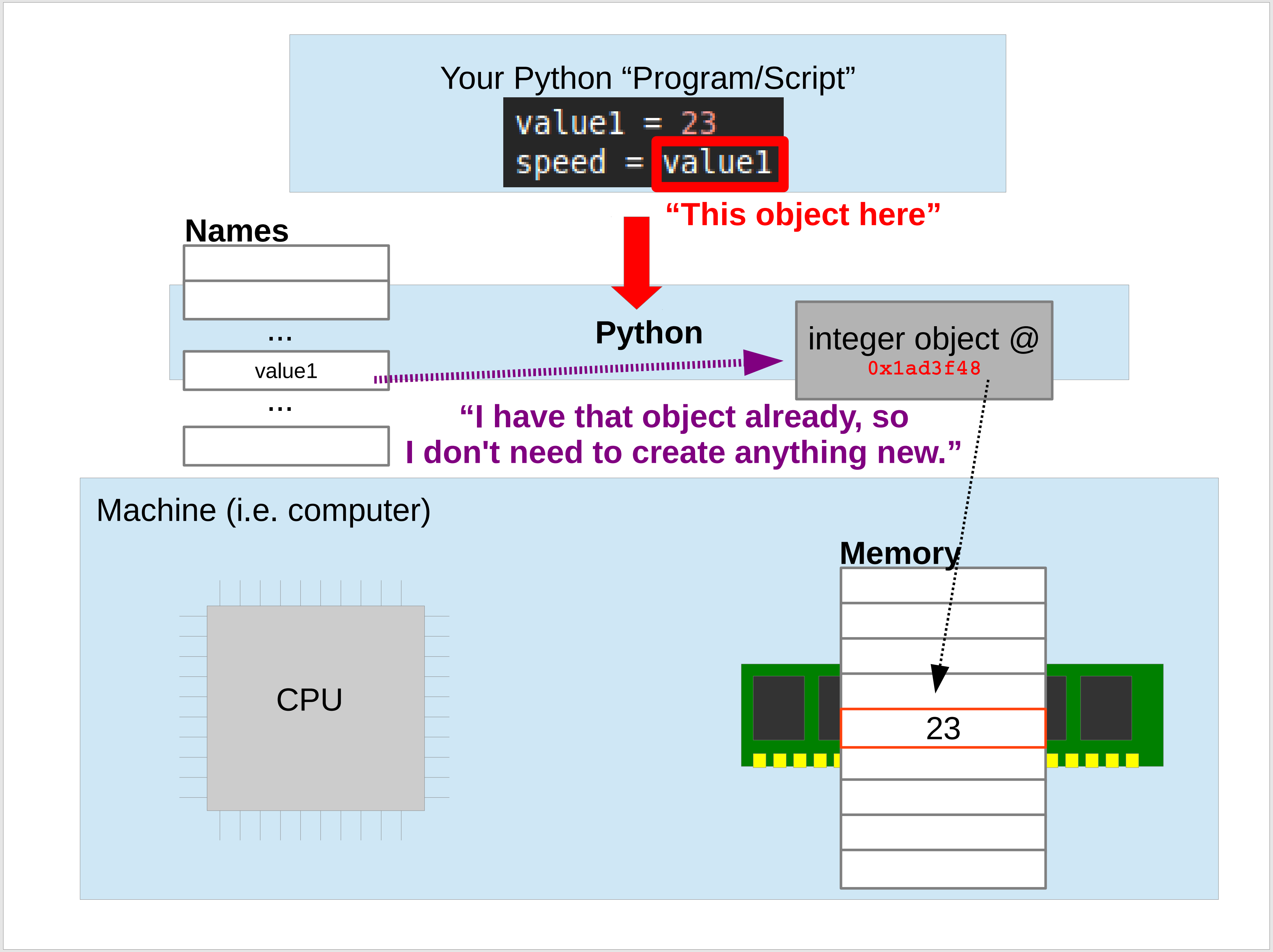
```
value1 = 23
speed = value1
```

The Python Execution Model



```
value1 = 23
speed = value1
```

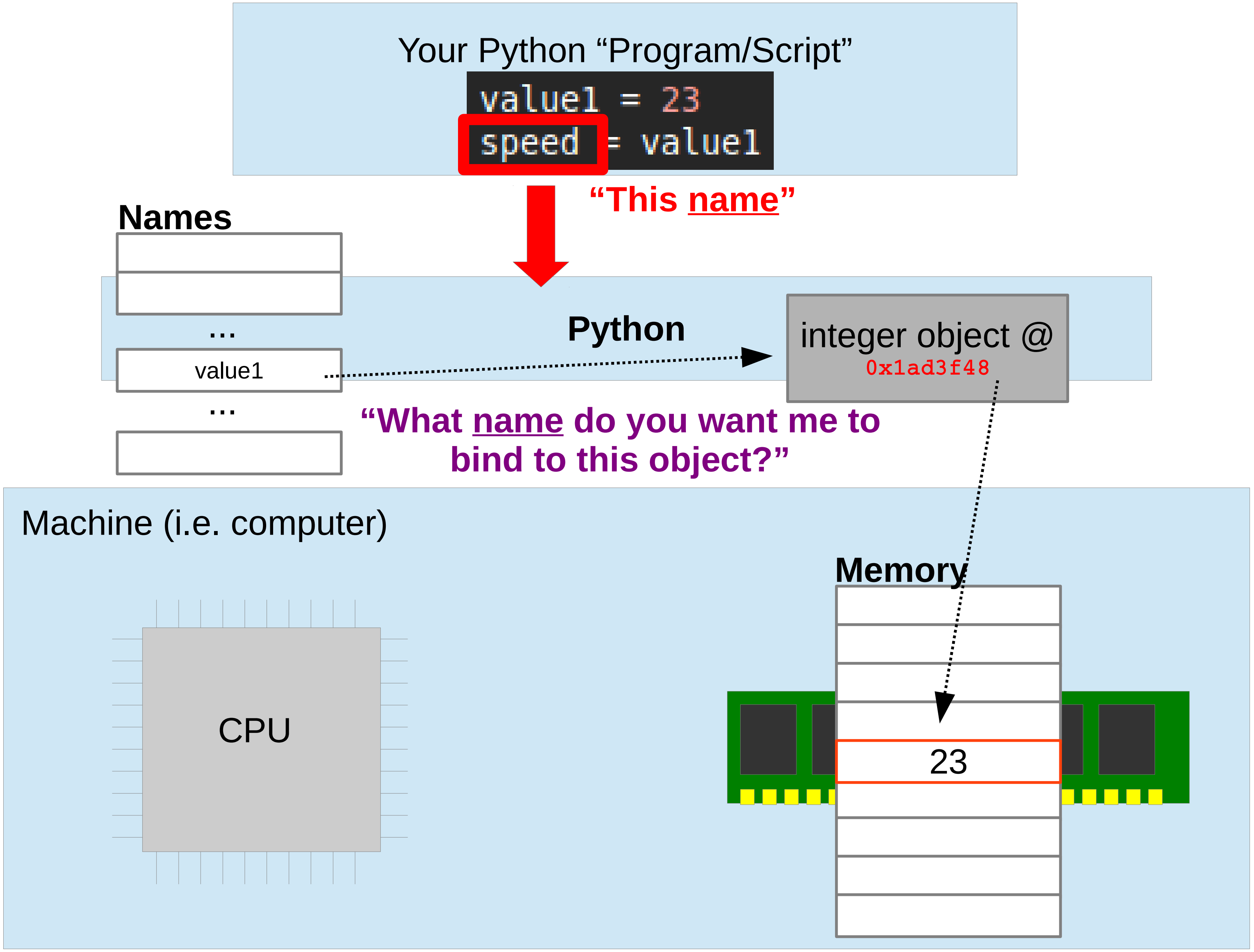
The Python Execution Model



Your Python "Program/Script"

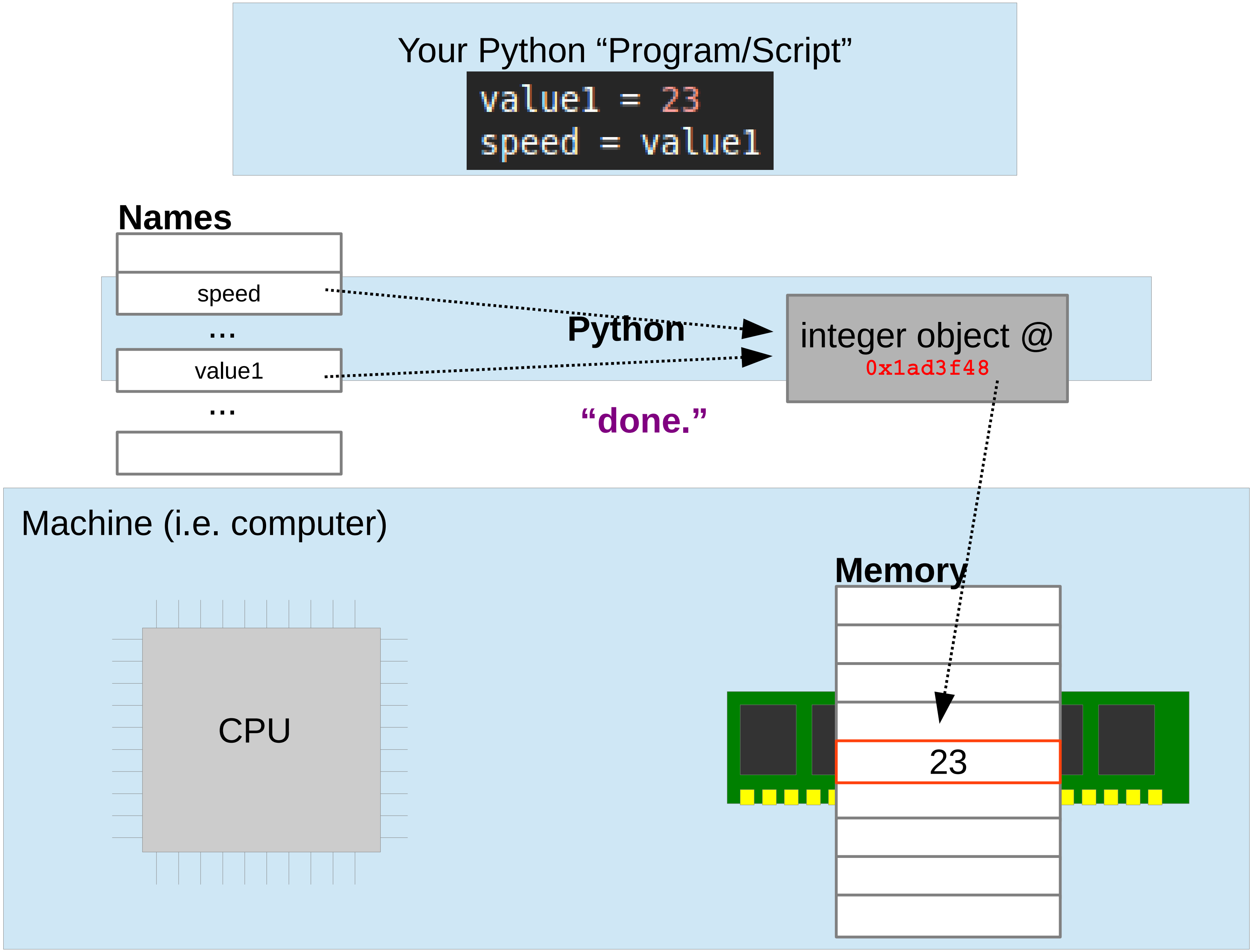
```
value1 = 23
speed = value1
```

The Python Execution Model



```
value1 = 23
speed = value1
```

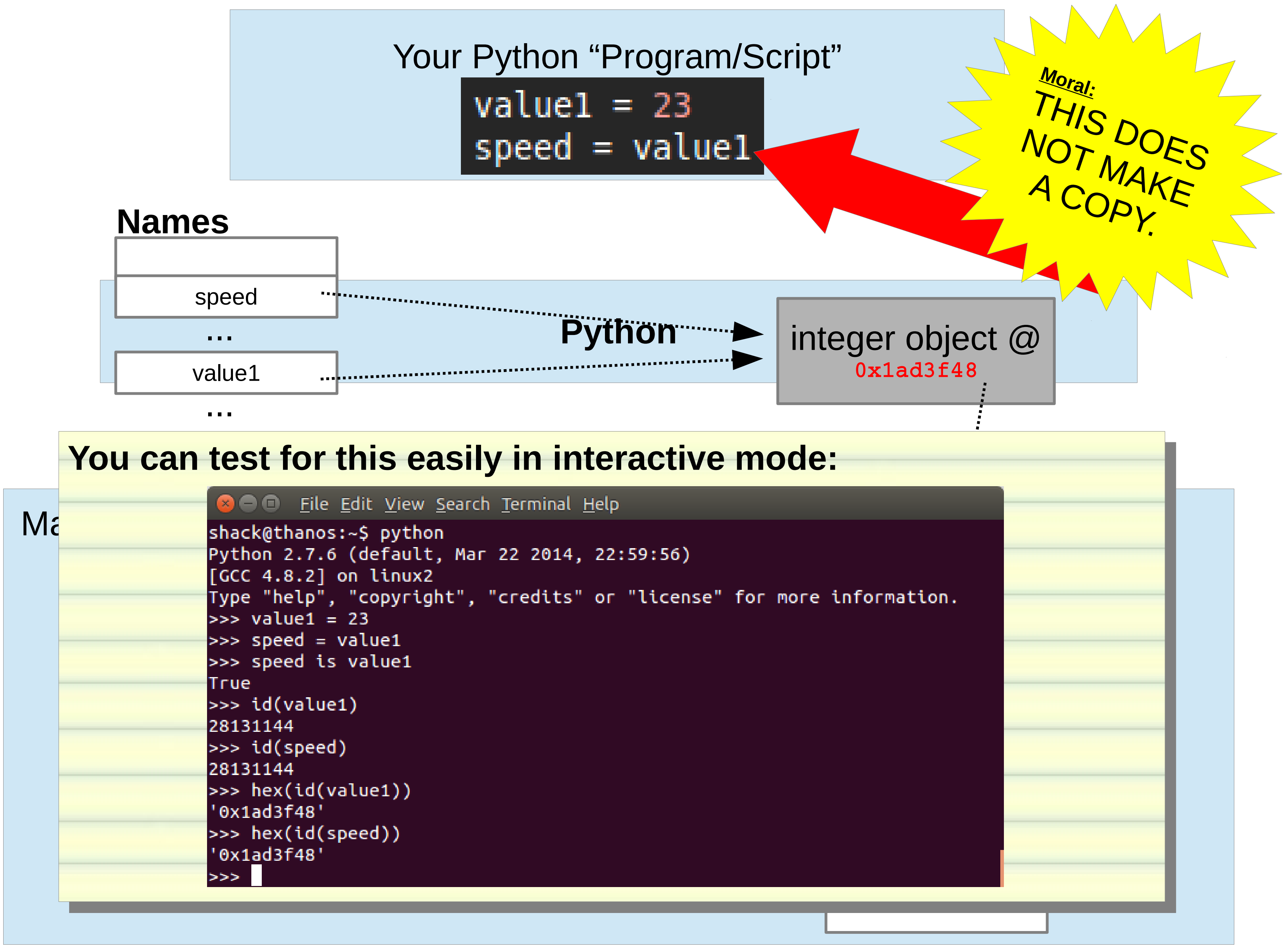
The Python Execution Model



```
value1 = 23
speed = value1
```

The Python Execution Model

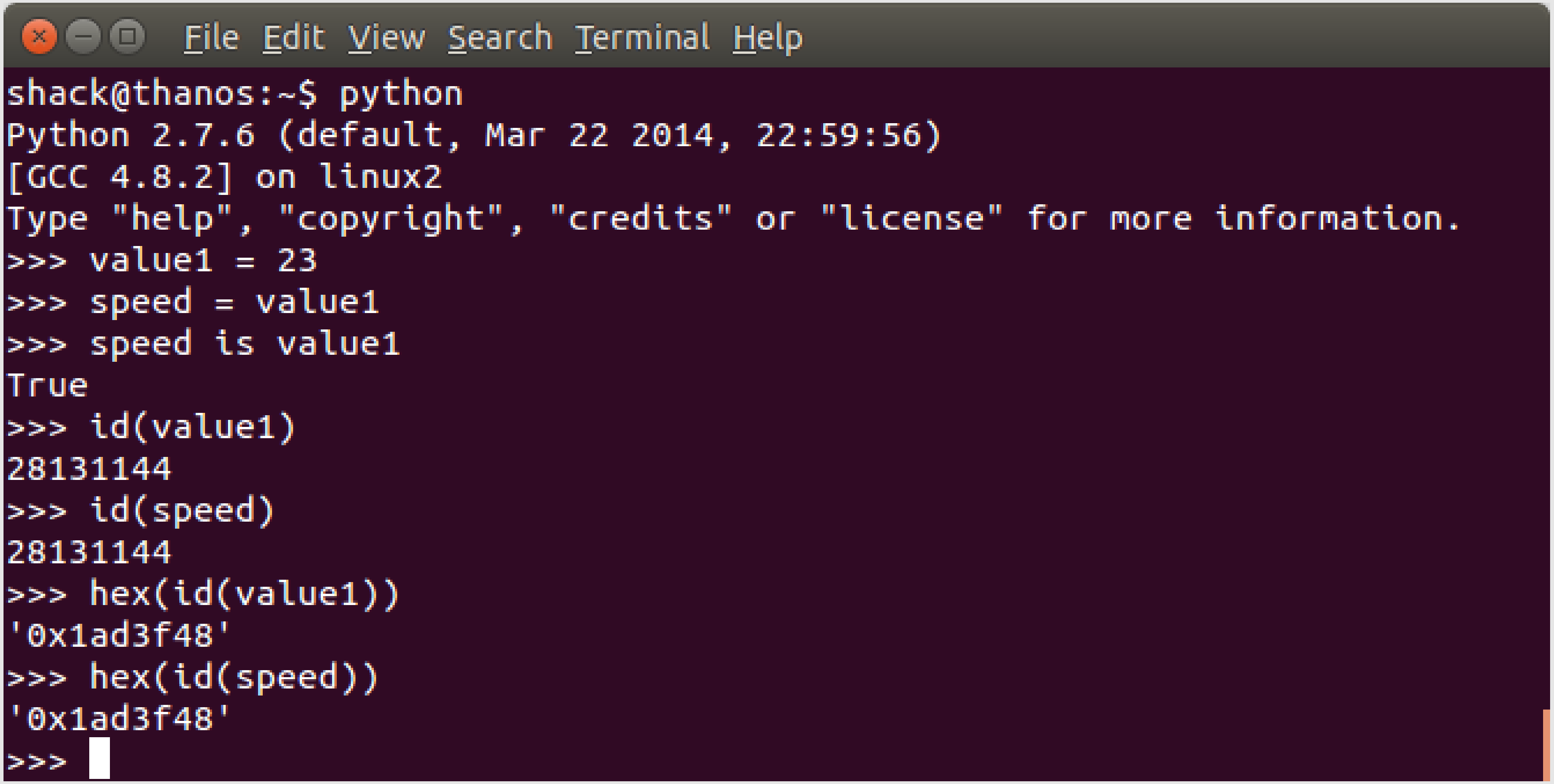
The Python Execution Model



You can test for this easily in interactive mode:

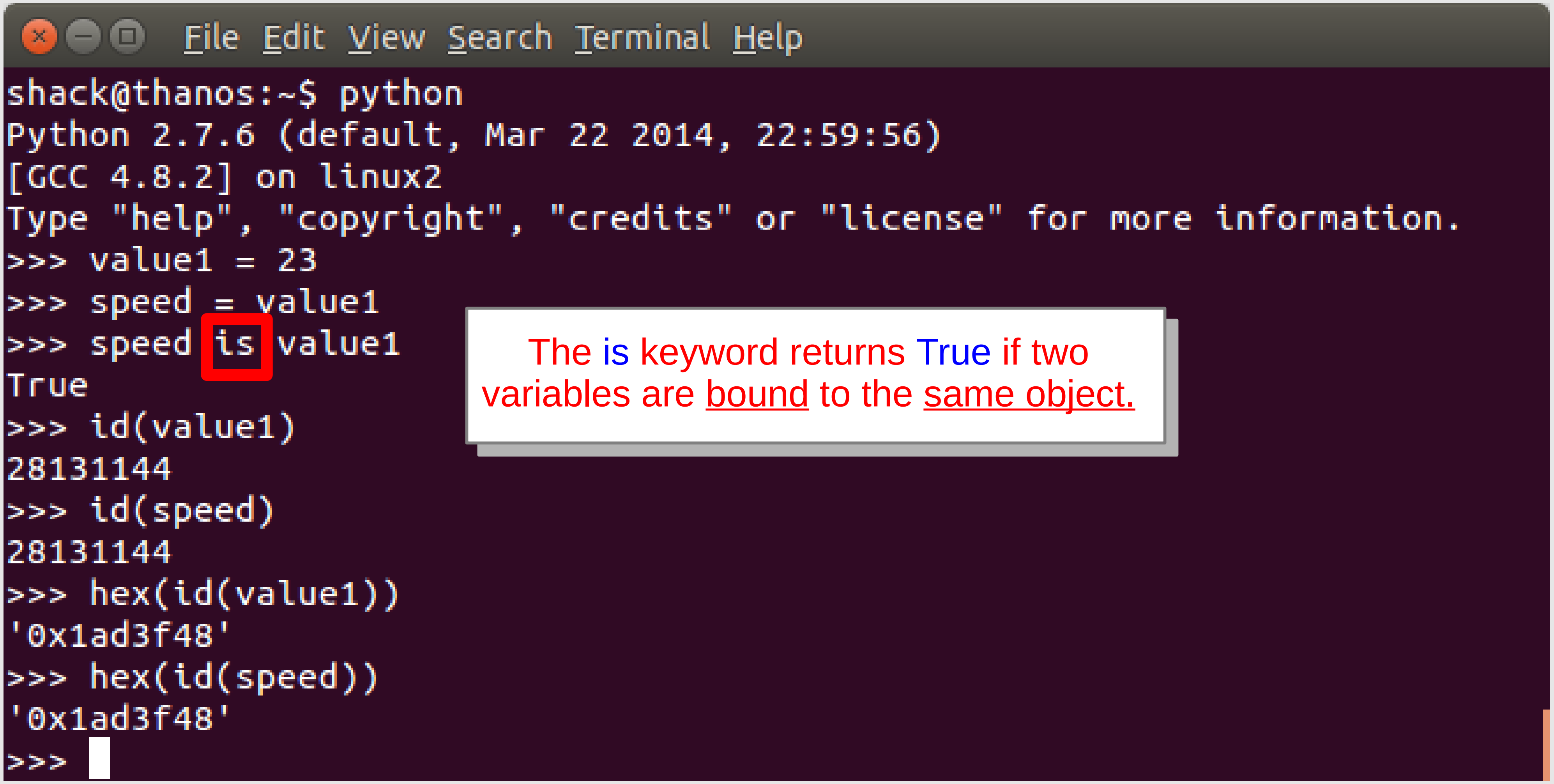
```
shack@thanos:~$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> value1 = 23
>>> speed = value1
>>> speed is value1
True
>>> id(value1)
28131144
>>> id(speed)
28131144
>>> hex(id(value1))
'0x1ad3f48'
>>> hex(id(speed))
'0x1ad3f48'
>>> 
```

The Python Execution Model



```
shack@thanos:~$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> value1 = 23
>>> speed = value1
>>> speed is value1
True
>>> id(value1)
28131144
>>> id(speed)
28131144
>>> hex(id(value1))
'0x1ad3f48'
>>> hex(id(speed))
'0x1ad3f48'
>>> 
```

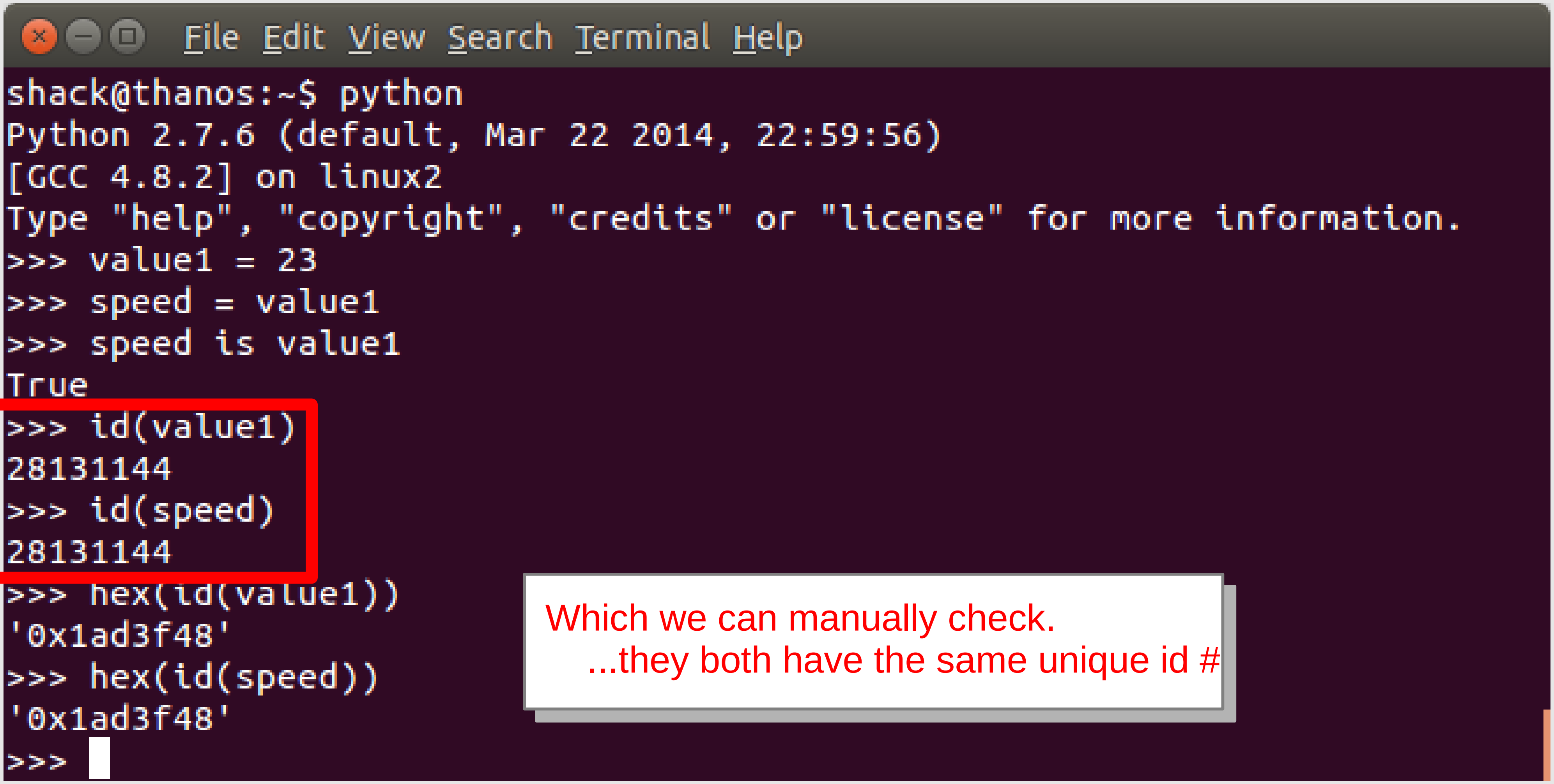
The Python Execution Model



```
shack@thanos:~$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> value1 = 23
>>> speed = value1
>>> speed is value1
True
>>> id(value1)
28131144
>>> id(speed)
28131144
>>> hex(id(value1))
'0x1ad3f48'
>>> hex(id(speed))
'0x1ad3f48'
>>> 
```

The is keyword returns True if two variables are bound to the same object.

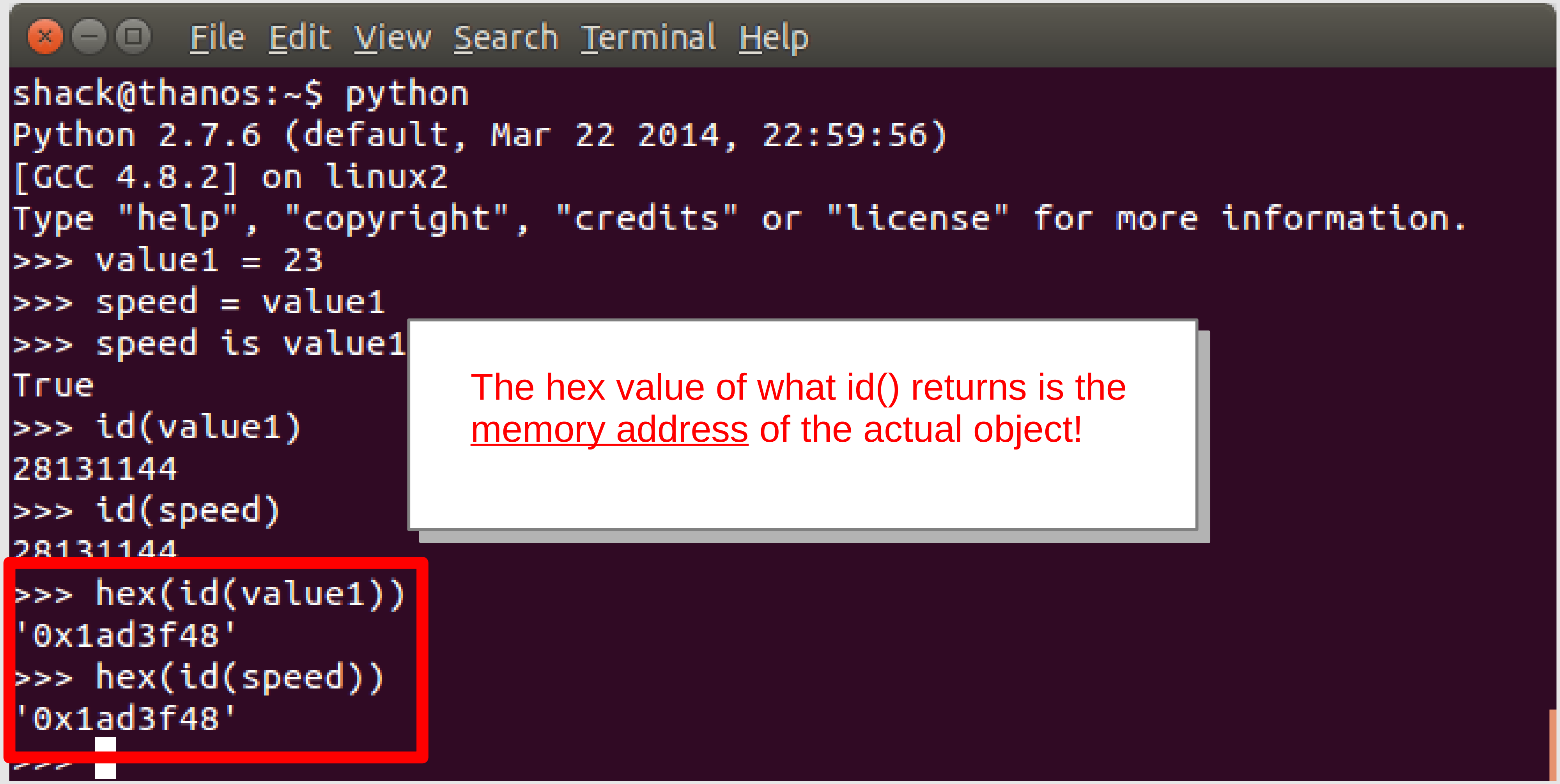
The Python Execution Model



```
shack@thanos:~$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> value1 = 23
>>> speed = value1
>>> speed is value1
True
>>> id(value1)
28131144
>>> id(speed)
28131144
>>> hex(id(value1))
'0x1ad3f48'
>>> hex(id(speed))
'0x1ad3f48'
>>> 
```

Which we can manually check.
...they both have the same unique id #

The Python Execution Model



```
shack@thanos:~$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> value1 = 23
>>> speed = value1
>>> speed is value1
True
>>> id(value1)
28131144
>>> id(speed)
28131144
>>> hex(id(value1))
'0x1ad3f48'
>>> hex(id(speed))
'0x1ad3f48'
>>> 
```

The hex value of what id() returns is the memory address of the actual object!

The Python Execution Model

Okay, Shackleford…

Still not convinced this important!

Well… you don't know about **lists**… or **dictionaries**… or anything ***mutable***.

(yet.)

The Python Execution Model

Okay, Shackleford…

Still not convinced this important!

huh?!

Well… you don't know about **lists**… or **dictionaries**… or anything ***mutable***.

(yet.)

Fundamental Datatypes

Mutable (adj.) – State ***can*** be changed after creation.

Immutable (adj.) – State **cannot** be changed after creation.

Mutable Python Types

- **list**

  Similar to a vector in MATLAB, but not confined to just numbers. Can also be heterogeneous!

  example:
  ```
  >>> A = [3.24, 78, 'foo', 1103]
  >>> A[1:3]
  [78, 'foo']
  ```

- **dictionary**

  An associative array.

  example:
  ```
  >>> A = {'age': 34, 'gender': 'female'}
  >>> A['gender']
  'female'
  ```

Immutable Python Types

- **int, float, long, complex**

- **tuple**

  Similar to a **list**, but values cannot be changed after creation. Consequently, a bit faster.

  example:
  ```
  >>> A = (32, 'bar', 32.22)
  >>> A[0:2]
  (32, 'bar')
  ```

- **str**

  A string of characters

  example:
  ```
  >>> A = "Hello World!"
  >>> A[3:9]
  'lo Wor'
  ```

Fundamental Datatypes

Mutable (adj.) – State ***can*** be changed after creation.

Immutable (adj.) – State **cannot** be changed after creation.

Mutable Python Types

- **list**

  Similar to a vector in MATLAB, but not confined to just numbers. Can also be heterogeneous!

  example:
```
>>> A = [3.24, 78, 'foo
>>> A[1:3]
[78, 'foo']
```

- **dictionary**

  An associative array.

  example:
```
>>> A = {'age': 34, 'gender': 'female'}
>>> A['gender']
'female'
```

Immutable Python Types

- **int, float, long, complex**
- **tuple**

  Similar to a **list**, but values cannot be changed after
  quently, a bit faster.

```
'bar', 32.22)
```

  A string of characters

  example:
```
>>> A = "Hello World!"
>>> A[3:9]
'lo Wor'
```

DON'T PANIC!

There will be other lectures.

We will cover these in more depth.

Fundamental Datatypes

Mutable (adj.) – State ***can*** be changed after creation.

Immutable (adj.) – State **cannot** be changed after creation.

Mutable Python Types

- **list**
- **(**

Immutable Python Types

- **int, float, long, complex**

t values cannot be changed after
ently, a bit faster.

```
bar', 32.22)
```

rs

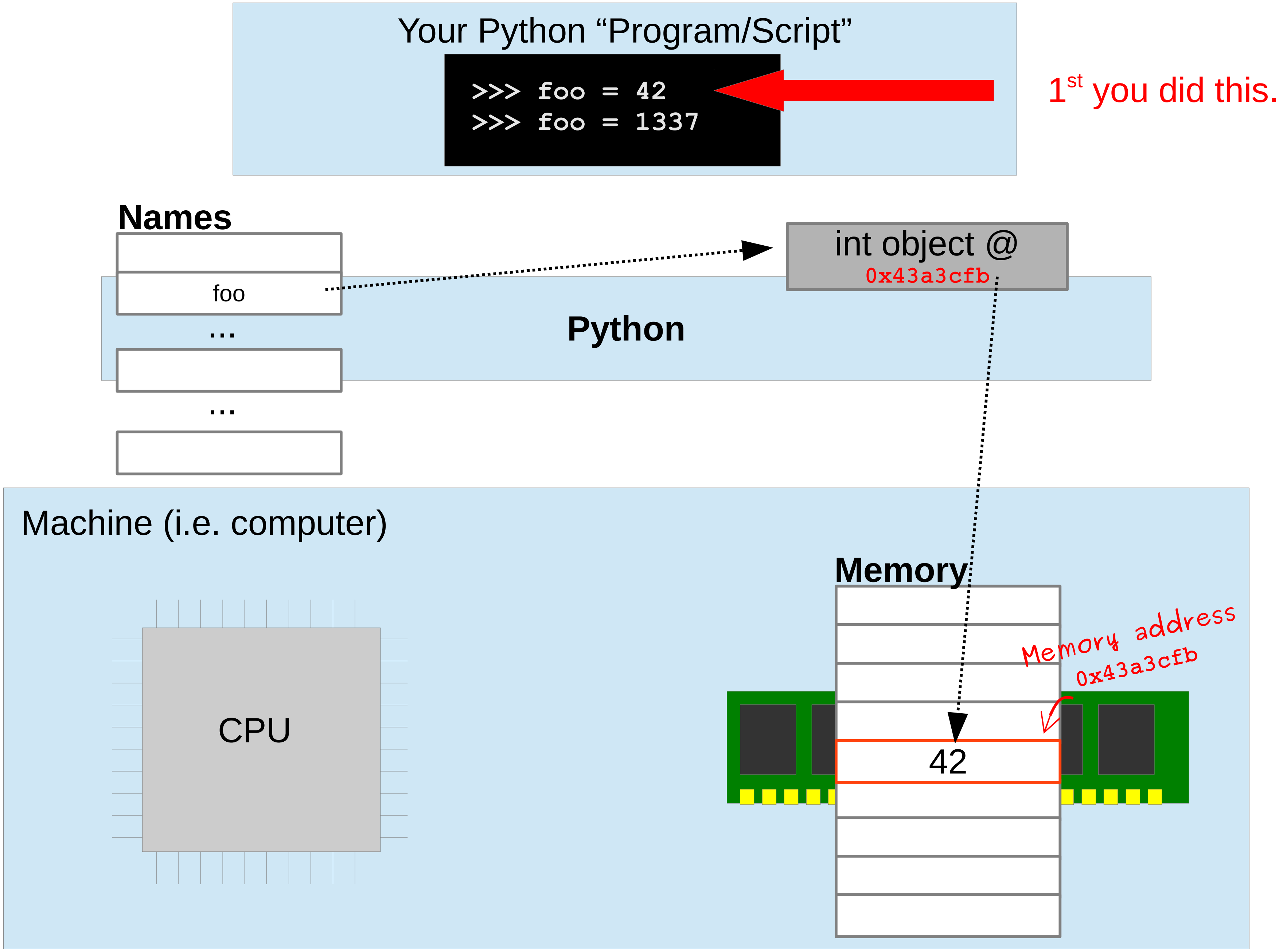
```
World!"
```

"Hold up, Shack. How is int immutable?
I can totally do *this* and change it!"

```
>>> foo = 42
>>> print foo
42
>>> foo = 1337
>>> print foo
1337
```

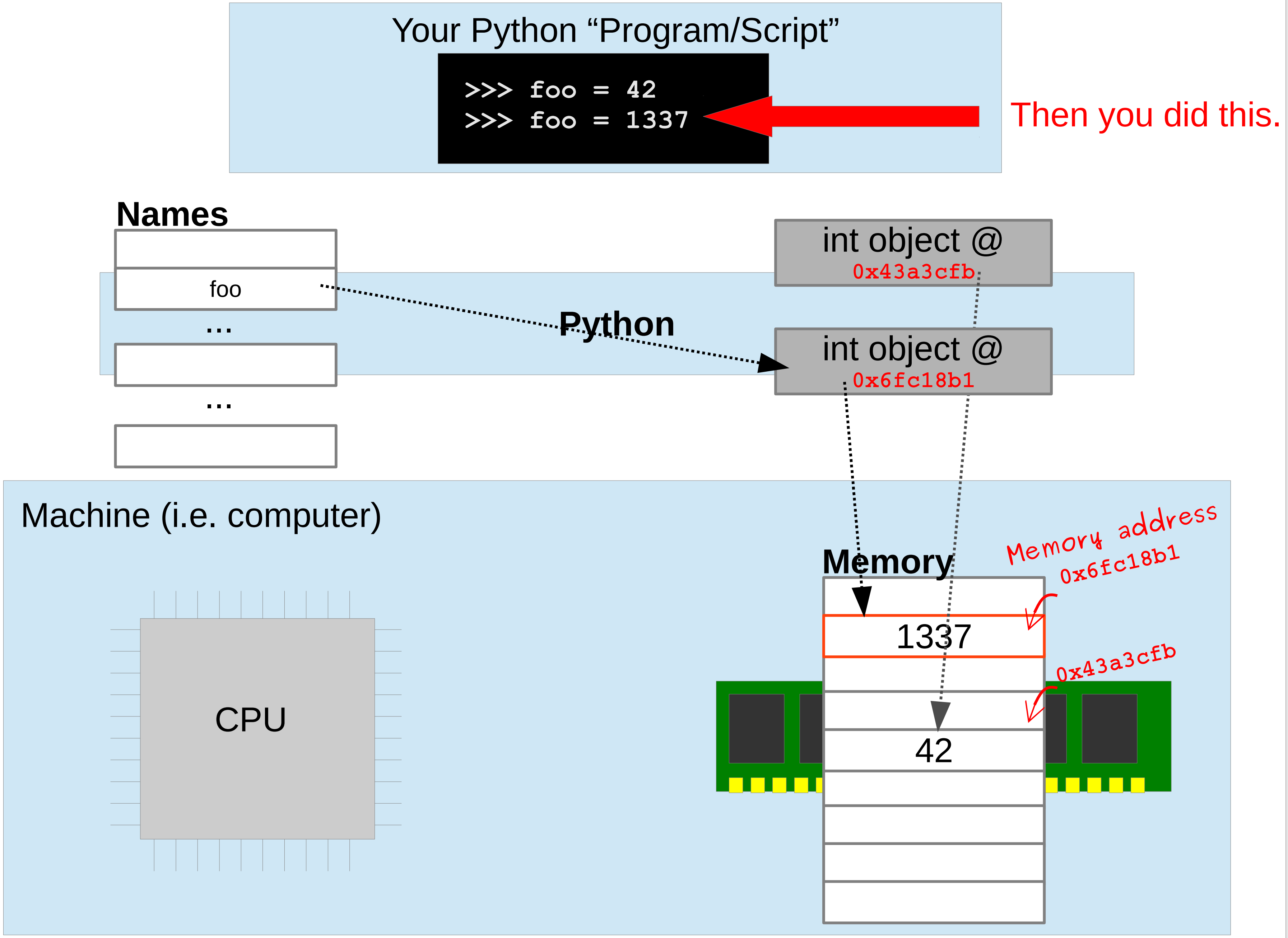
...no, you didn't change the int object storing 42. You did this:

The Python Execution Model



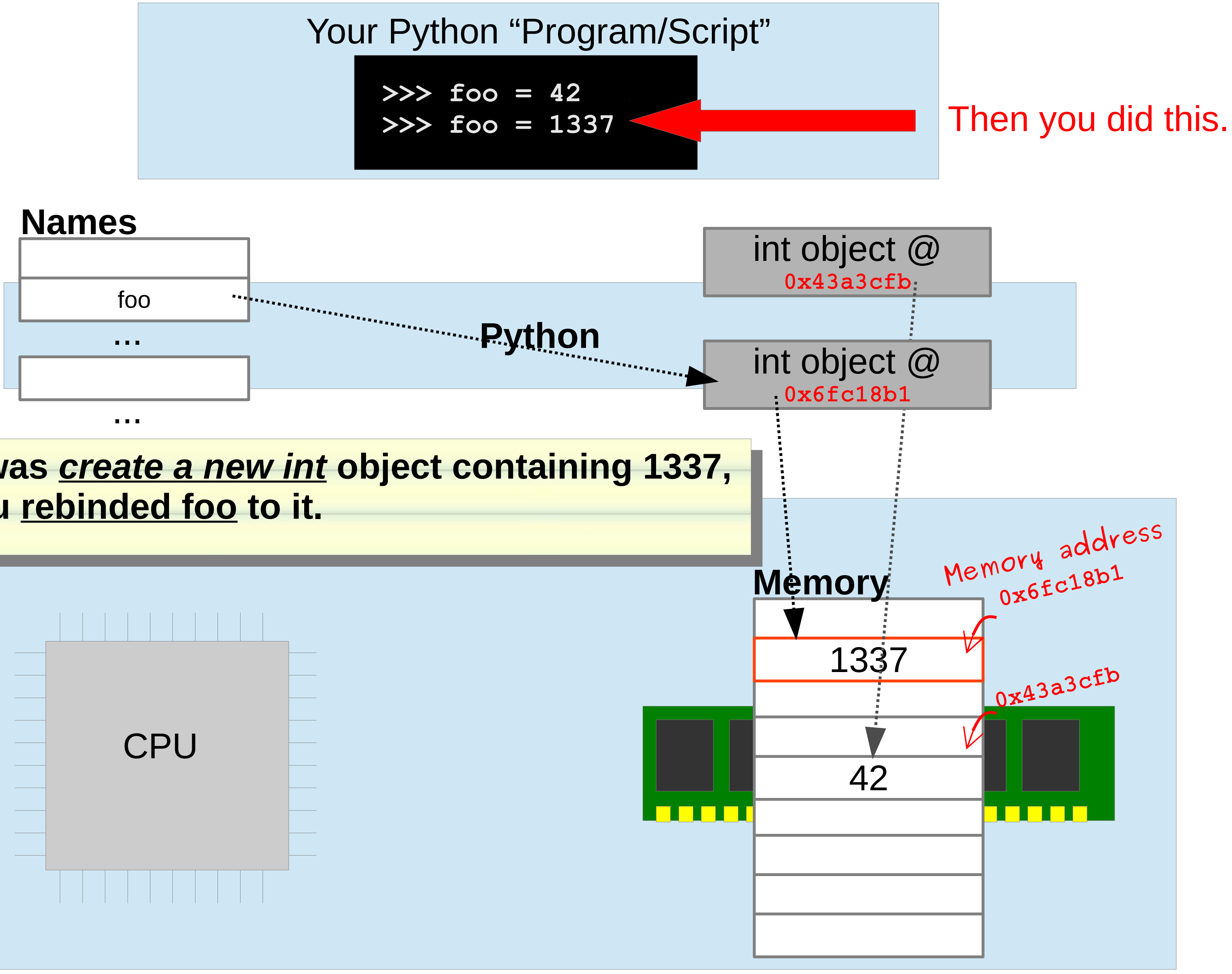
```
>>> foo = 42
>>> foo = 1337
```

The Python Execution Model



```
>>> foo = 42
>>> foo = 1337
```

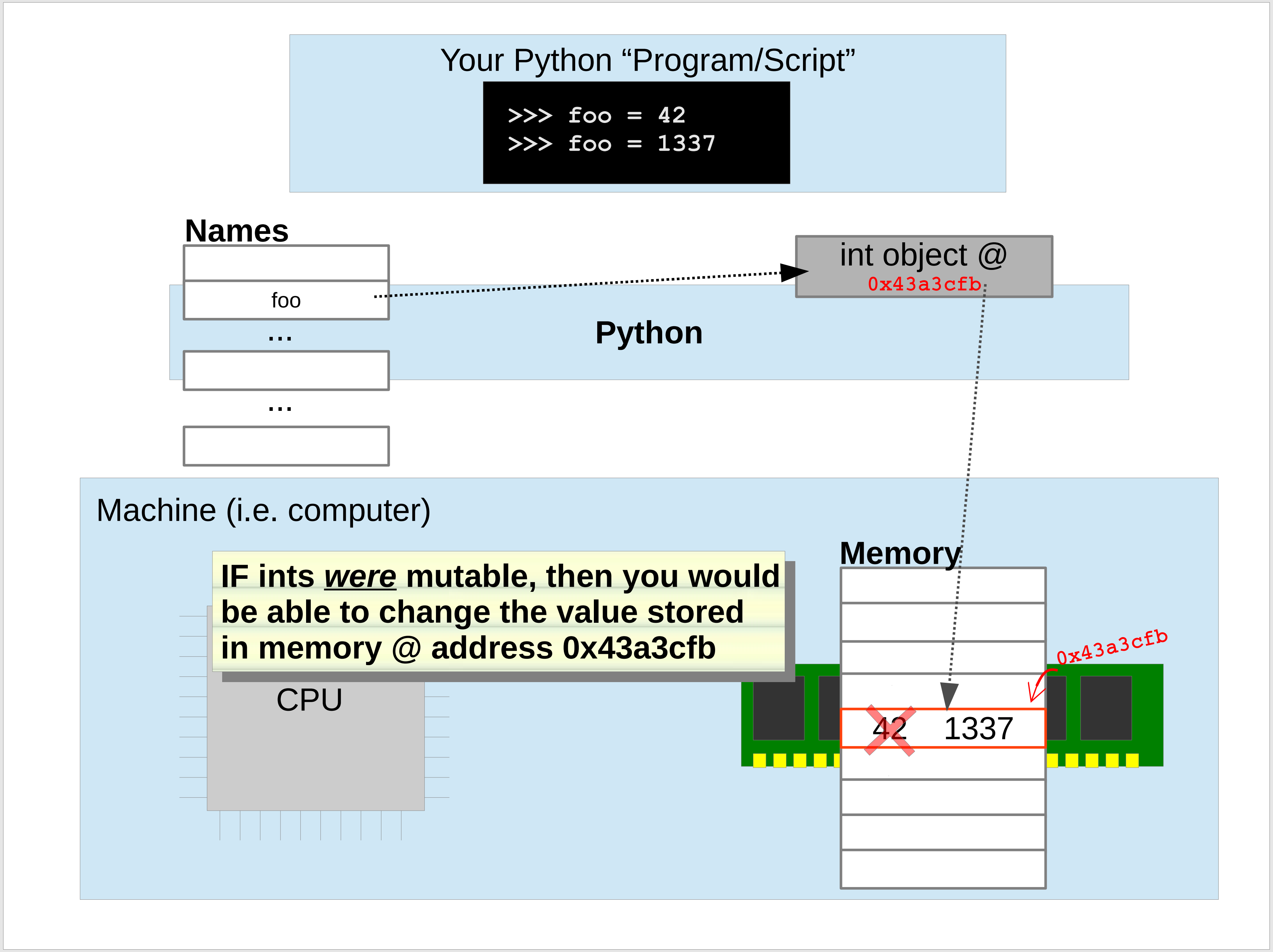
The Python Execution Model



```
>>> foo = 42
>>> foo = 1337
```

All you did was *create a new int* object containing 1337, and *then* you rebinded foo to it.

The Python Execution Model



```
>>> foo = 42
>>> foo = 1337
```

IF ints *were* mutable, then you would be able to change the value stored in memory @ address 0x43a3cfb

The Python Execution Model

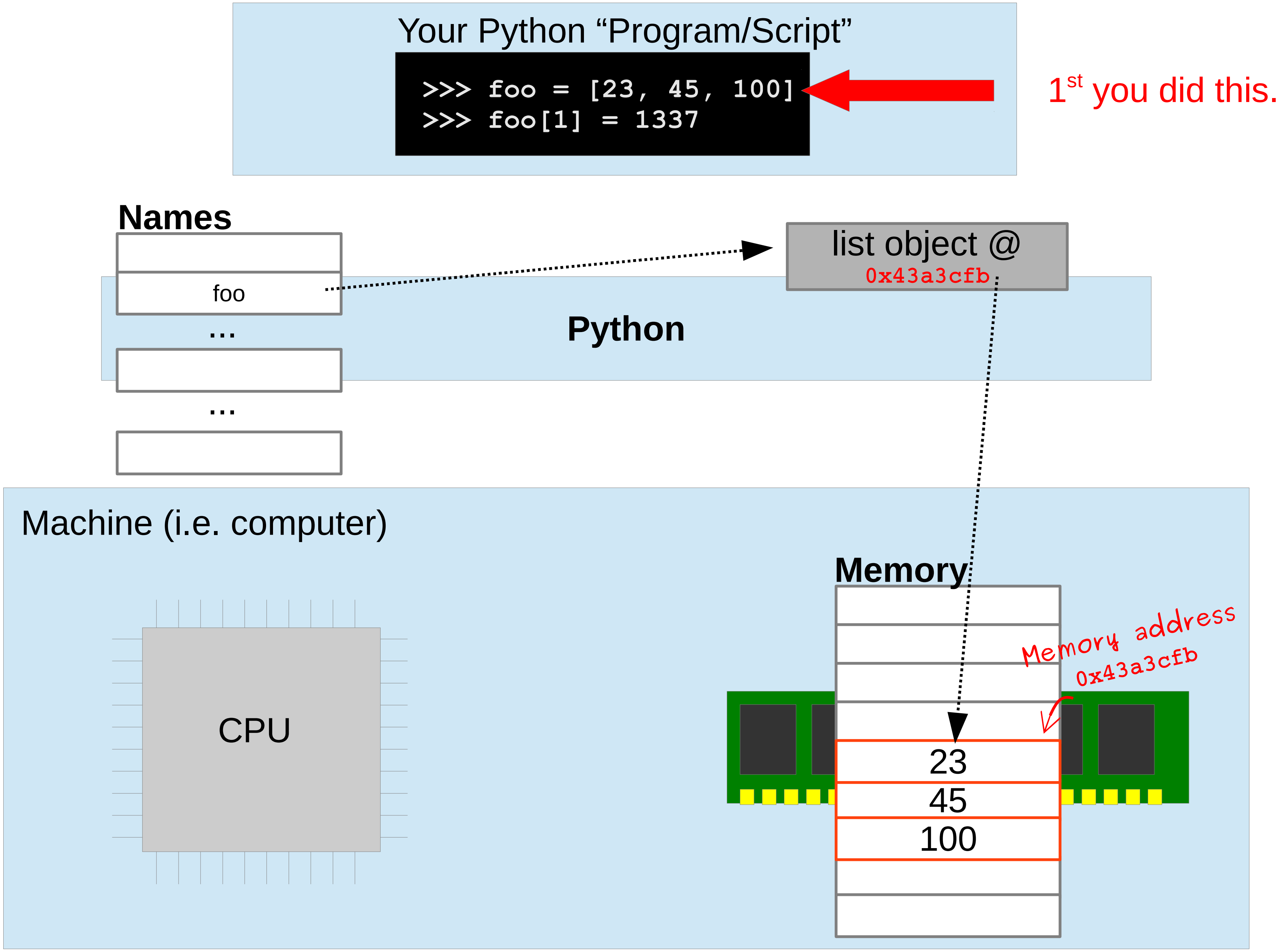
Okay, Shackleford…

Please, please, please…
why is this important?!

Well… let's look at a **list**.

```
>>> foo = [23, 45, 100]
>>> print foo
[23, 45, 100]
>>> foo[1] = 1337
>>> print foo
[23, 1337, 100]
```

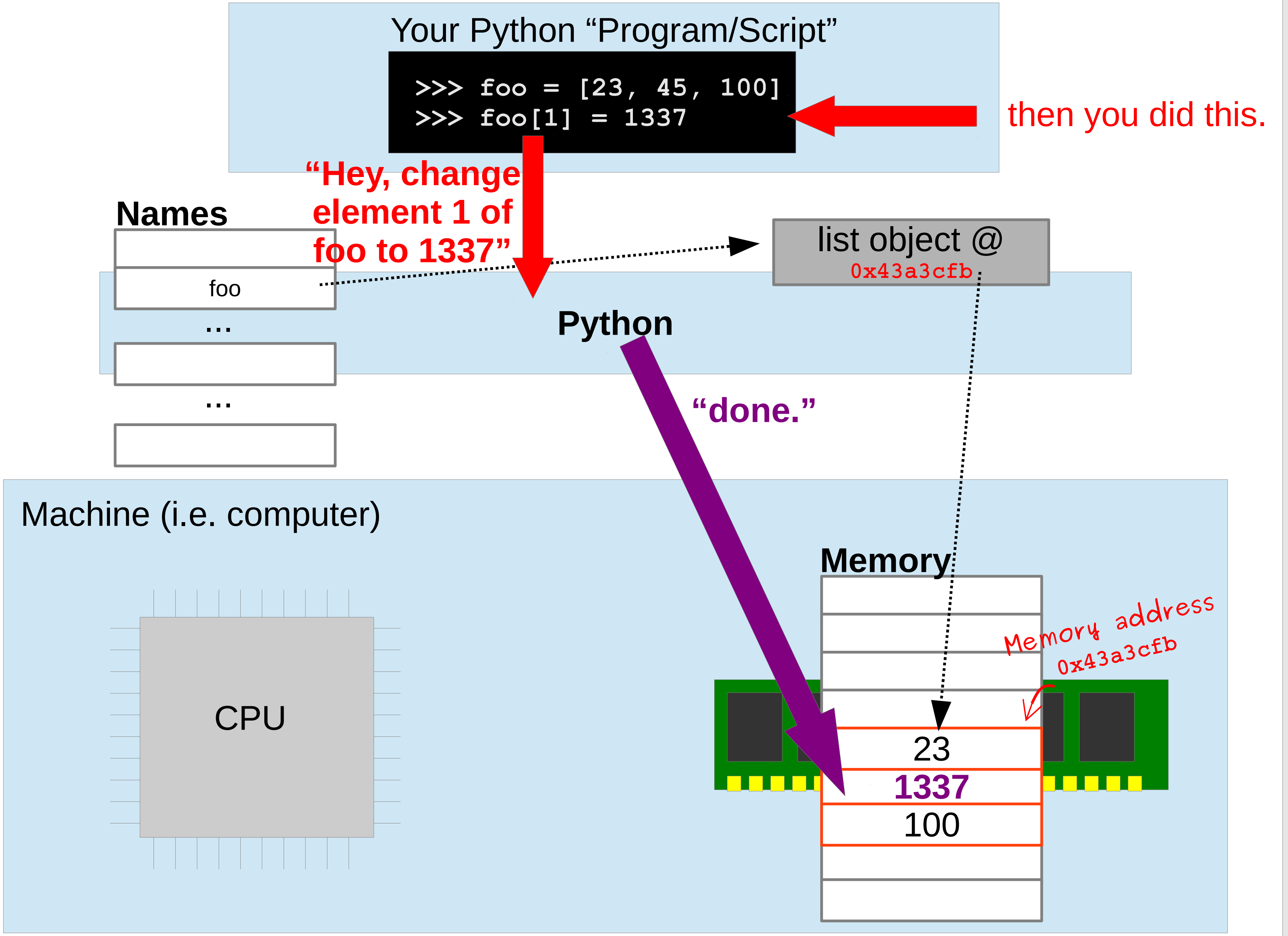
The Python Execution Model



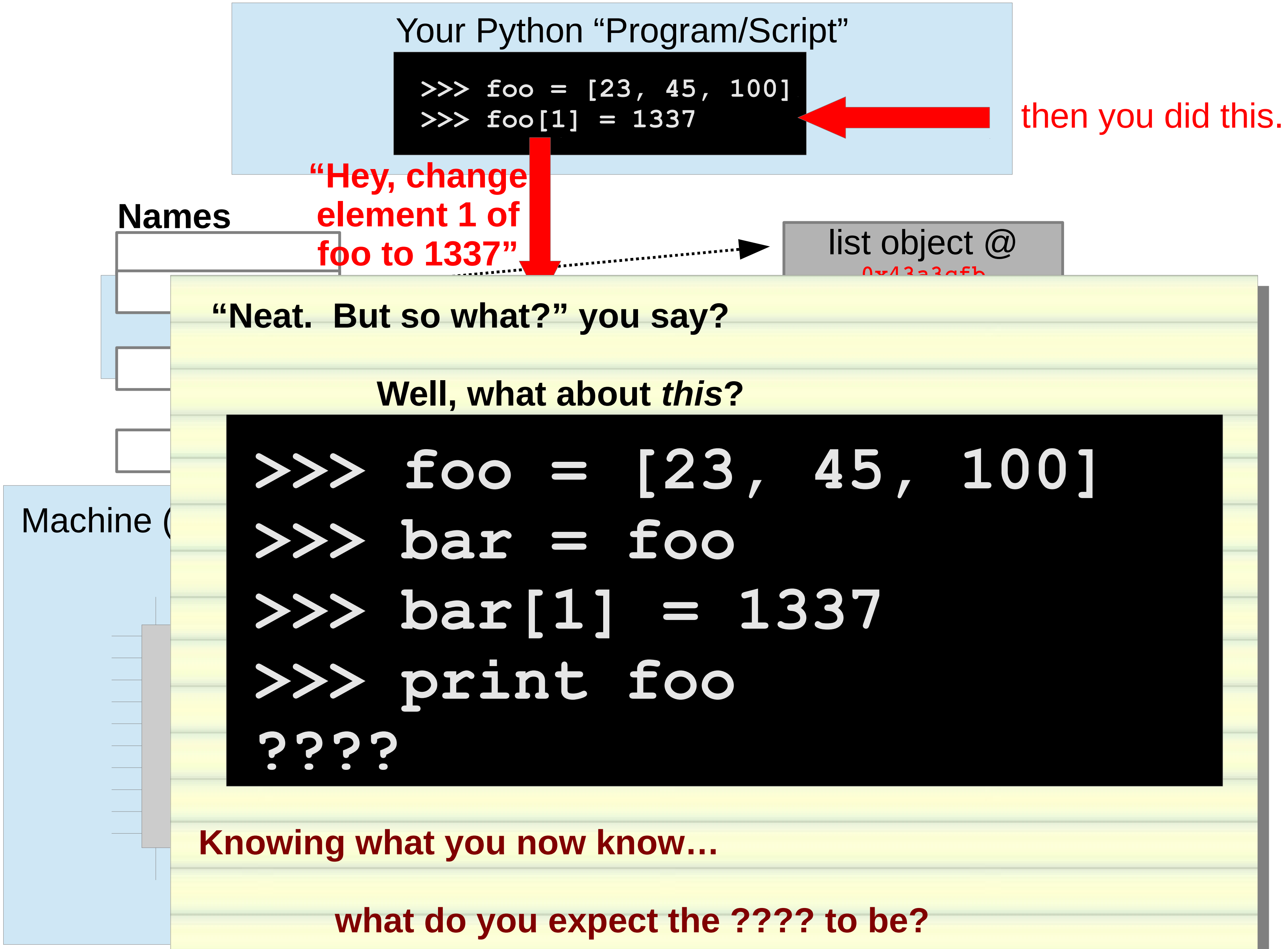
Your Python "Program/Script"

```
>>> foo = [23, 45, 100]
>>> foo[1] = 1337
```

The Python Execution Model


```
>>> foo = [23, 45, 100]
>>> foo[1] = 1337
```

The Python Execution Model



Your Python "Program/Script"
then you did this.
"Hey, change element 1 of foo to 1337"
Names
list object @
Machine (

```
>>> foo = [23, 45, 100]
>>> foo[1] = 1337
```

"Neat. But so what?" you say?

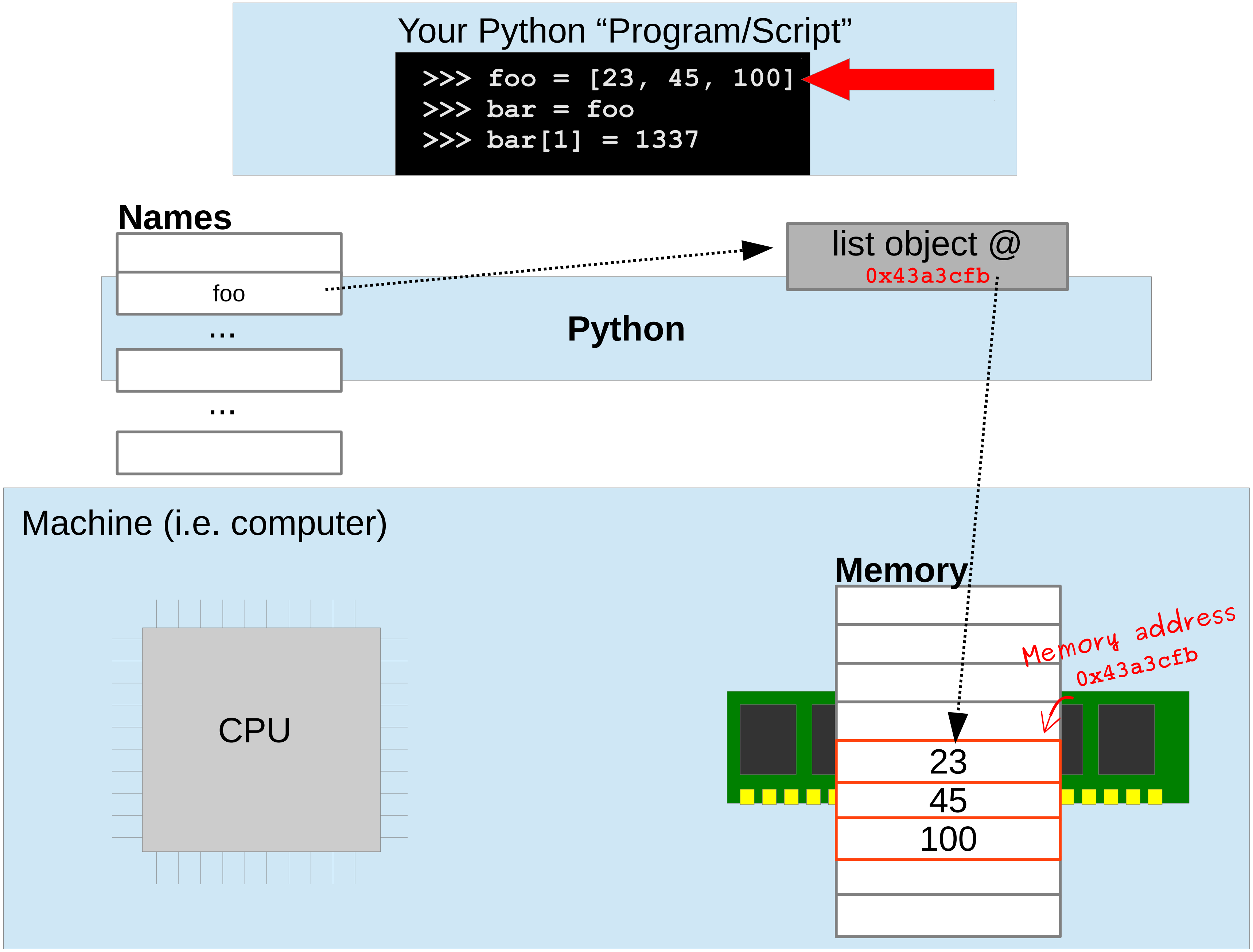
Well, what about *this*?

```
>>> foo = [23, 45, 100]
>>> bar = foo
>>> bar[1] = 1337
>>> print foo
????
```

Knowing what you now know…

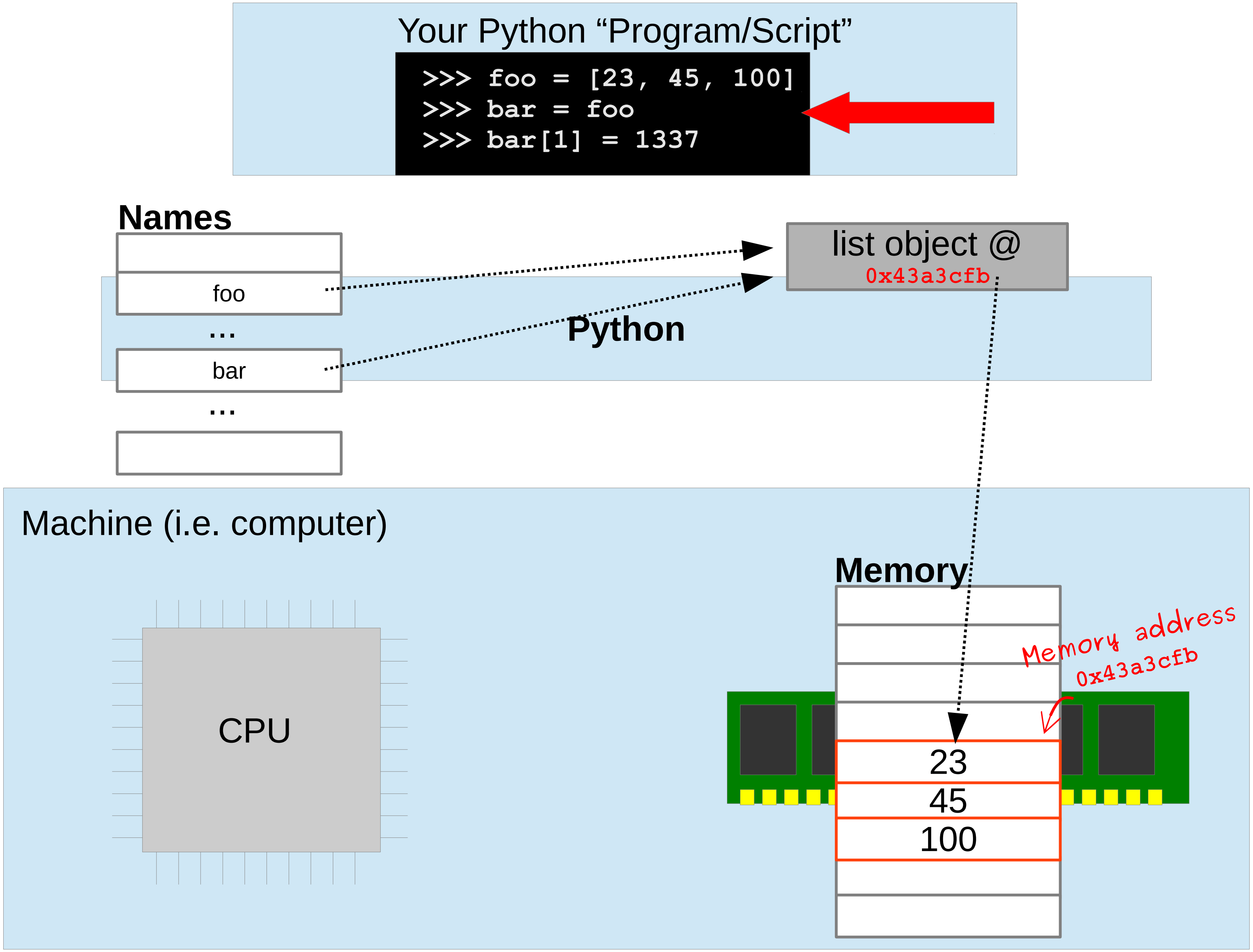
what do you expect the ???? to be?

The Python Execution Model



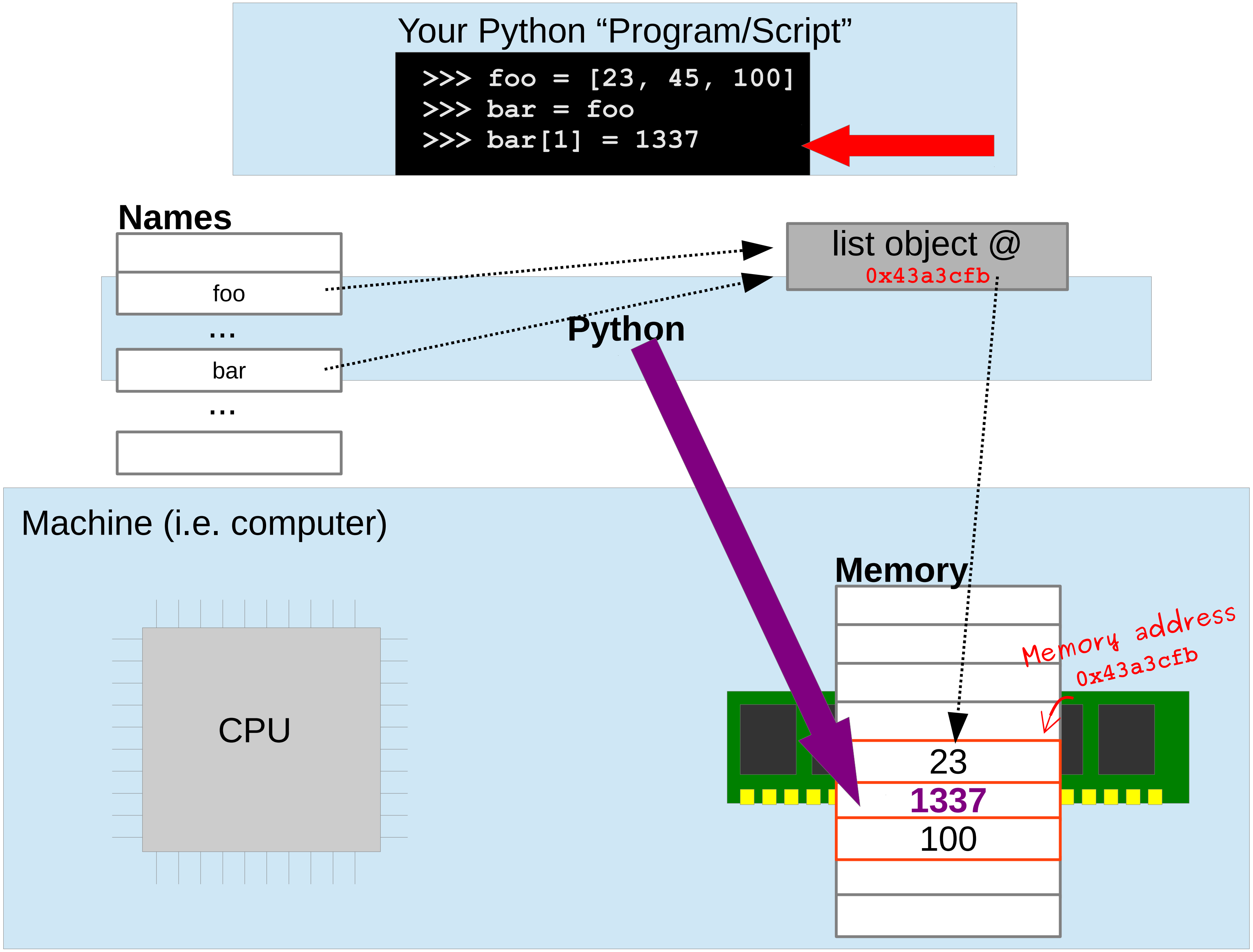
```
>>> foo = [23, 45, 100]
>>> bar = foo
>>> bar[1] = 1337
```

The Python Execution Model



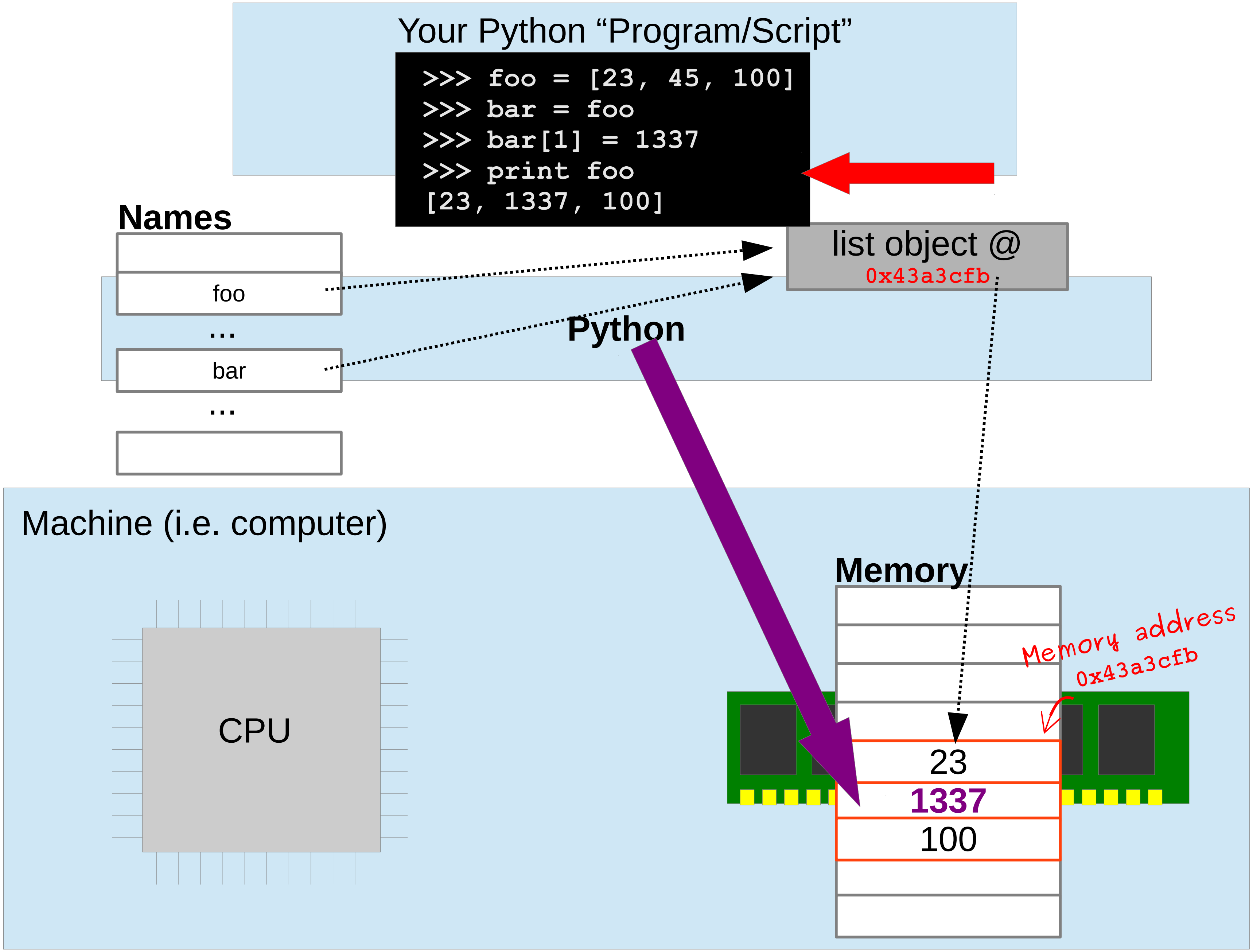
```
>>> foo = [23, 45, 100]
>>> bar = foo
>>> bar[1] = 1337
```

The Python Execution Model


Your Python “Program/Script”

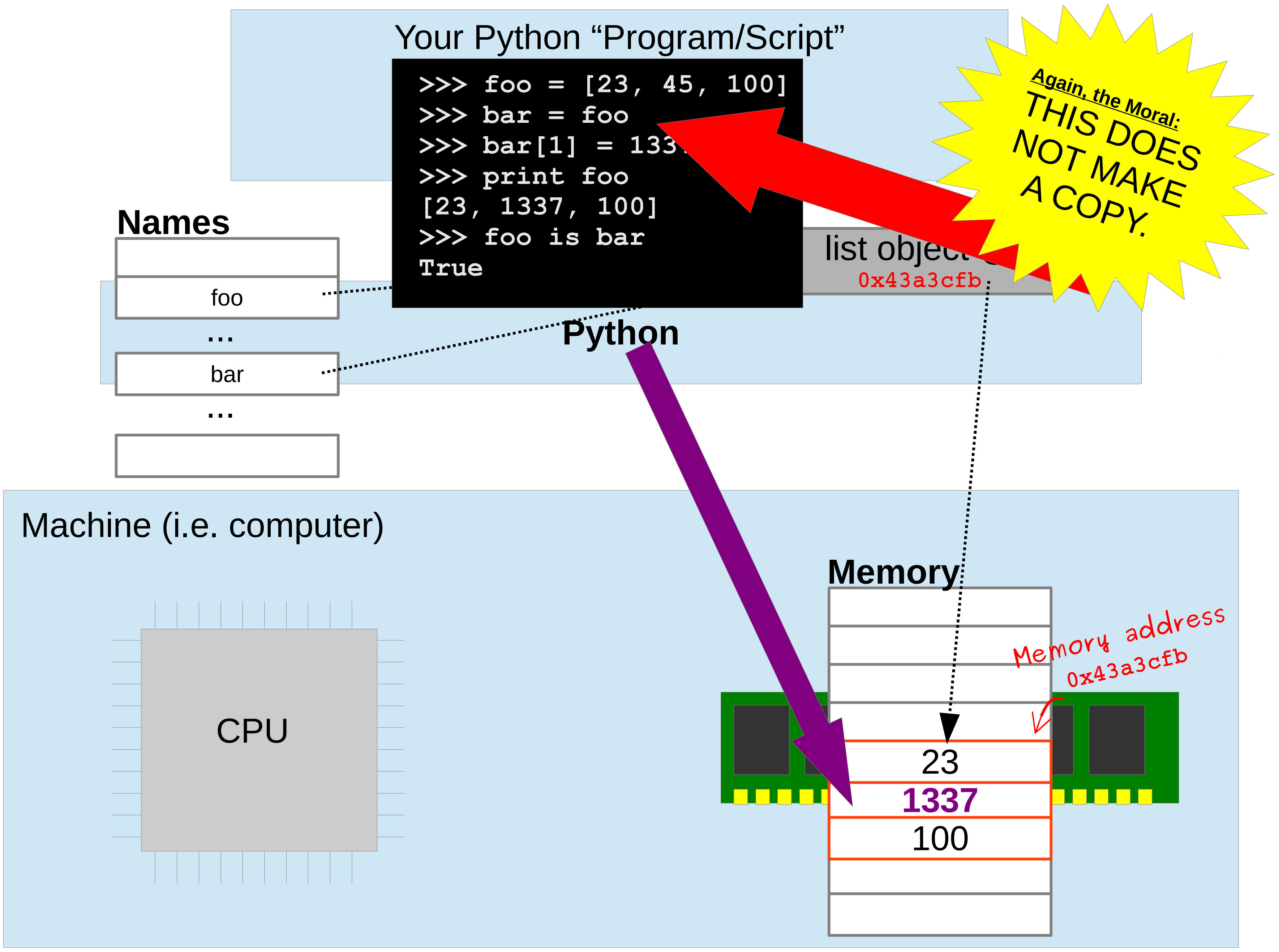
```
>>> foo = [23, 45, 100]
>>> bar = foo
>>> bar[1] = 1337
```

The Python Execution Model



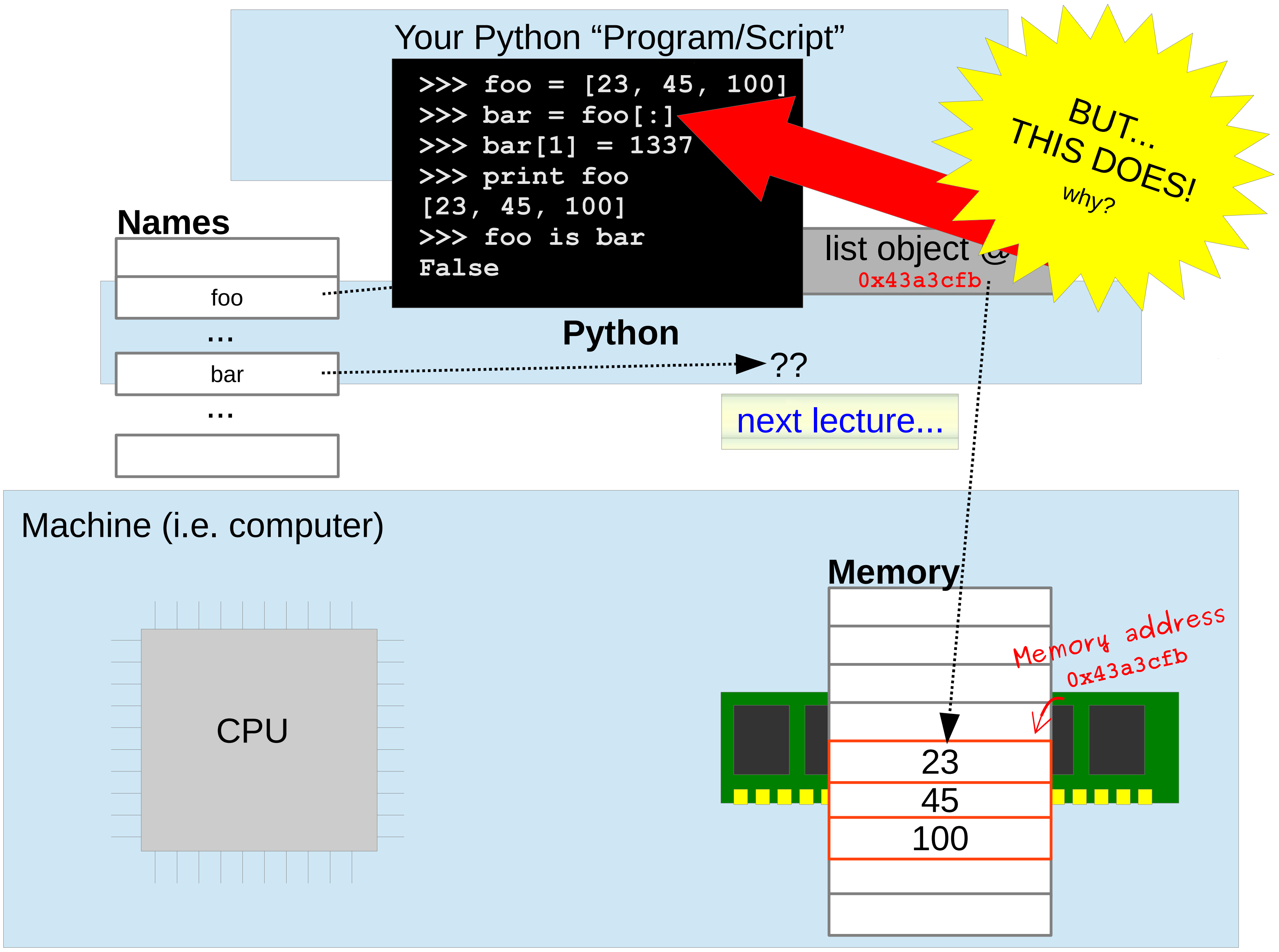
```
>>> foo = [23, 45, 100]
>>> bar = foo
>>> bar[1] = 1337
>>> print foo
[23, 1337, 100]
```

The Python Execution Model



```
>>> foo = [23, 45, 100]
>>> bar = foo
>>> bar[1] = 133
>>> print foo
[23, 1337, 100]
>>> foo is bar
True
```

The Python Execution Model



```
>>> foo = [23, 45, 100]
>>> bar = foo[:]
>>> bar[1] = 1337
>>> print foo
[23, 45, 100]
>>> foo is bar
False
```

Claim Your Accounts!

[1] You will receive an automated e-mail with your Thanos account info sometime today

[2] Login to the system via SSH like we discussed earlier. There is also a video tutorial on the Course website called
    “Logging into Thanos for the First Time”

[3] You will be forced to change your password ...to something good. Weak passwords will be rejected.

[4] You will be forced to logout

[5] Login with your new (STRONG) password and enjoy!

Claim Your Accounts!

...and finally…

IF YOU ARE **NOT REGISTERED** FOR THIS COURSE YET

YOU WILL NOT RECEIVE A THANOS ACCOUNT!

come see me now!