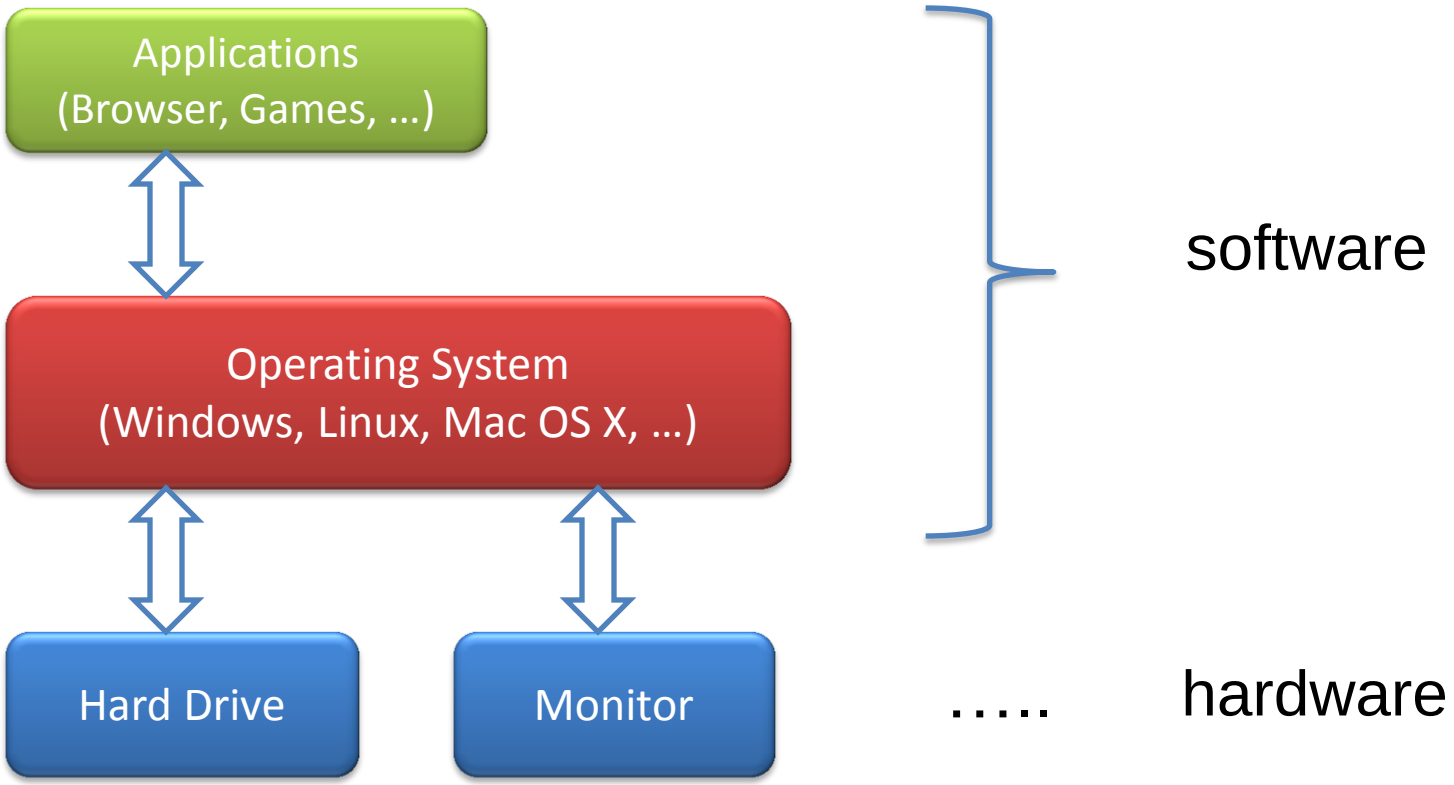
Announcements

Today you will write programs in the lab

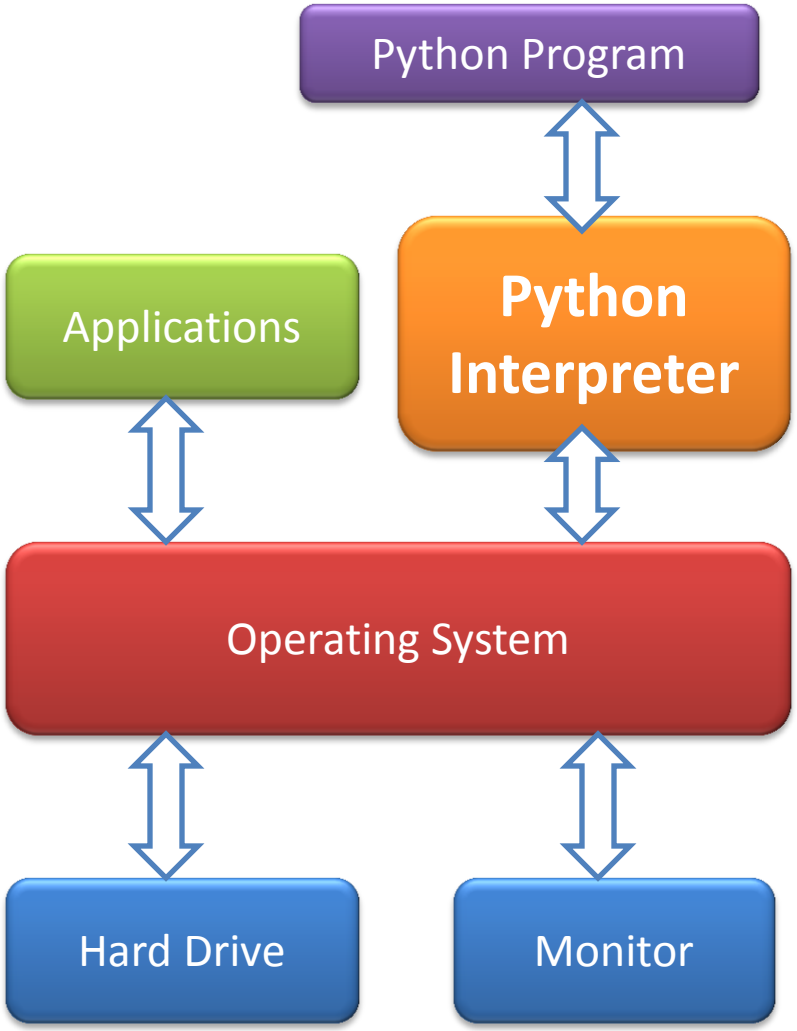
Programming Assignment → Thursday midnight.

TOWARDS PROGRAMMING WITH PYTHON

Hardware versus Software



Execution of Python Programs

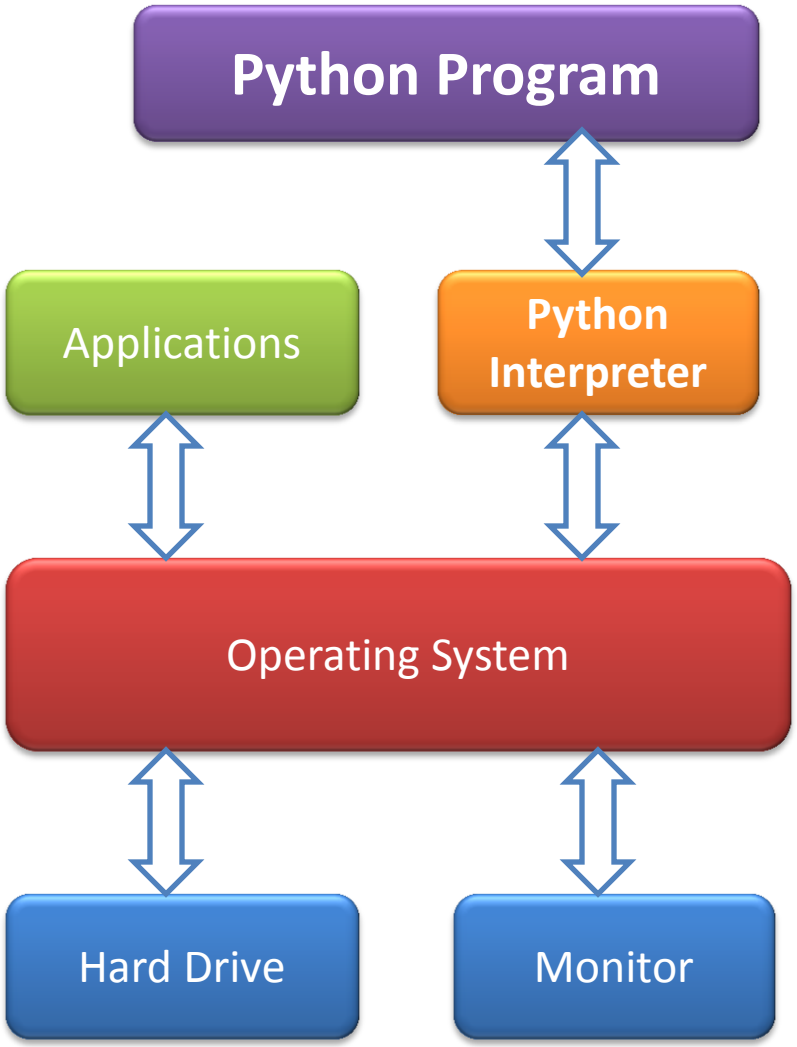


When you write a program in Python, Java etc. It does not run directly on the OS.

Another program called an interpreter or virtual machine takes it and runs it for you translating your commands into the language of the OS.

…

Execution of Python Programs



We will write Python programs that are executed by the Python Interpreter.

A Python Interpreter for your OS already exists. You can use the one on lab machines, install one for your laptop, or use one remotely

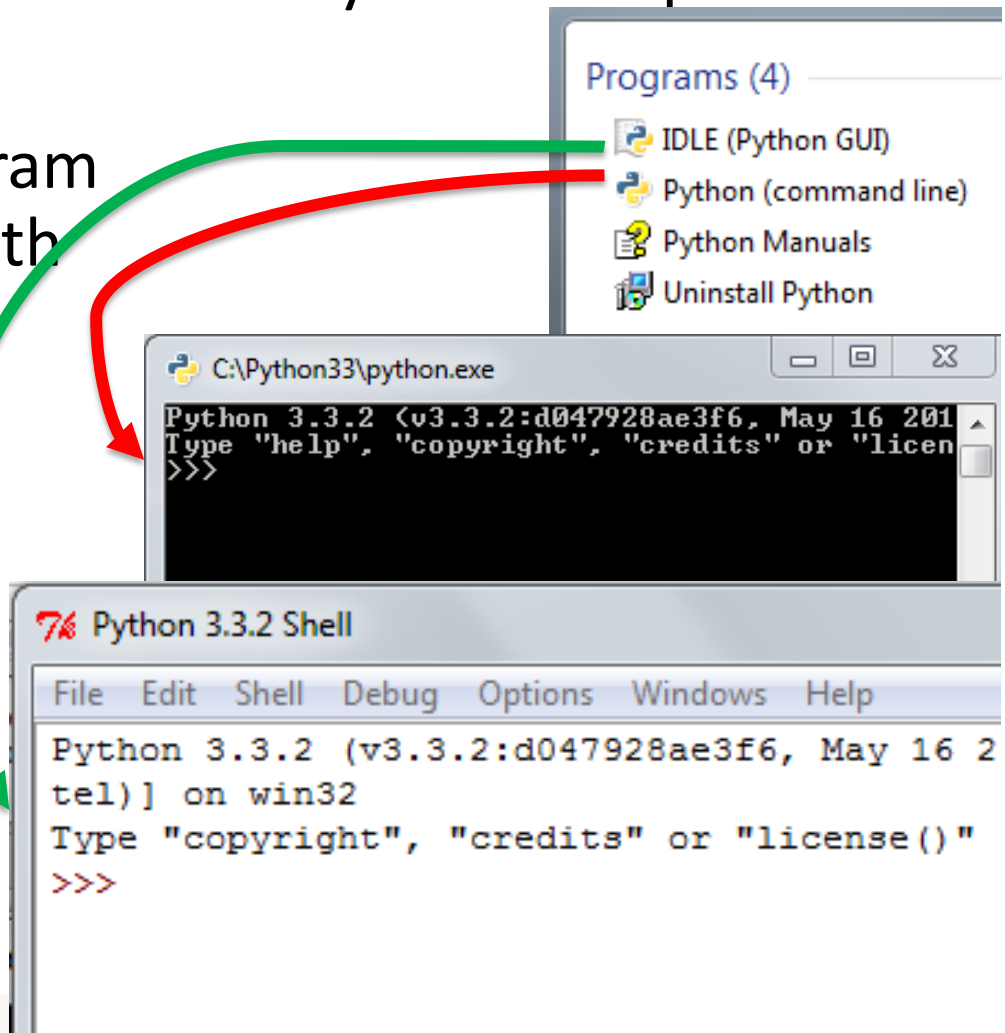
…

Using a Python Interpreter

There are two ways to interact with a Python interpreter:

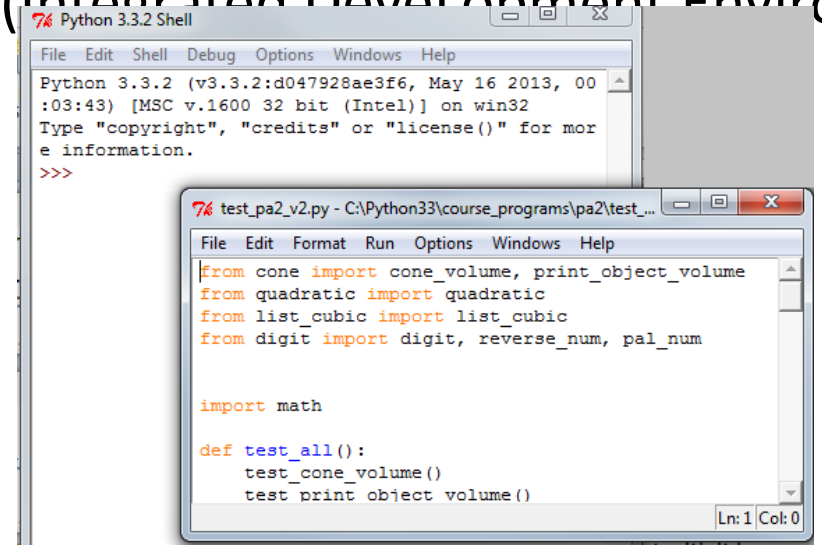
1. Tell it to execute a program that is saved in a file with a .py extension

2. Interact with it in a program called a **shell**

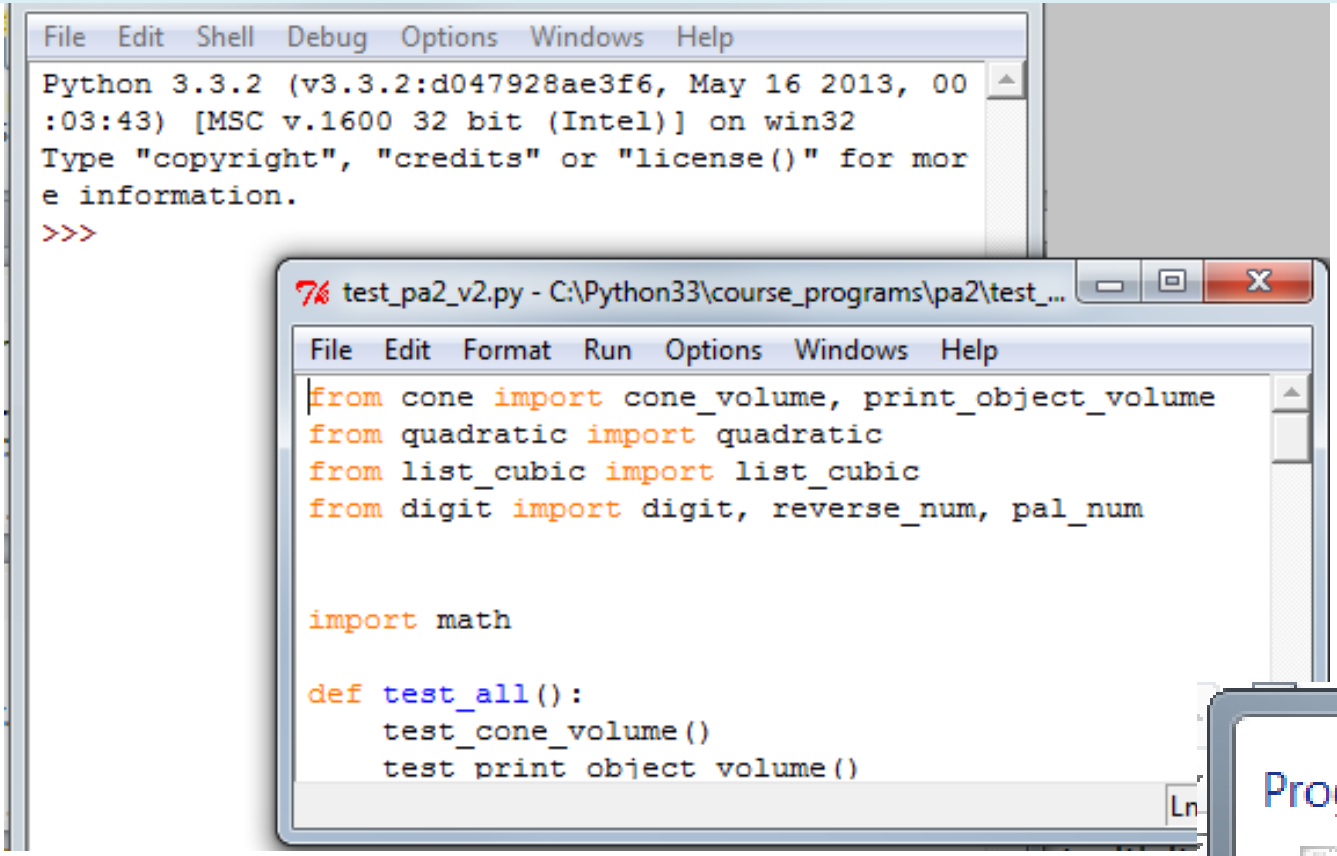


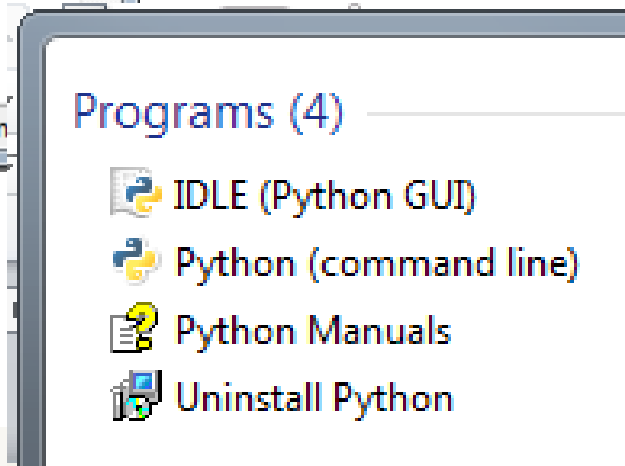
A Short Introduction

- Starting the Python interpreter either using remote access to a Unix Server at CMU or on your own computer
  - For specific instructions see the Resources page http://www.cs.cmu.edu/~15110-n15/resources.html
- Creating .py files with a text editor or
- Using IDLE (Integrated Development Environment)



Using IDLE

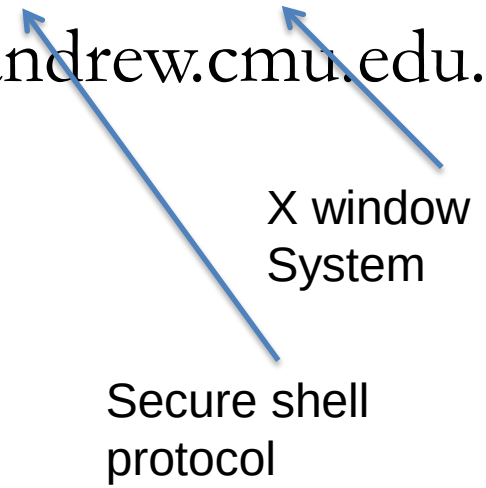




Idle3

Starting a Python Interpreter using Remote Access

- If you wish to work on your programming assignments from a physically remote location, we recommend that you use **ssh** and **X11** to run python3, gedit, etc., on unix.andrew.cmu.edu.



Using a Text Editor

- Files with the .py extension can be created by any editor but needs a Python interpreter to be read.

- We have chosen the integrated development environment of Python **(IDLE)** for the course but you may use an text editor of your own choice if you feel comfortable.
  - We suggest **gedit** as the editor but you may use an editor of your own choice if you feel comfortable.

Useful Unix Commands (Part 1)

All commands must be typed in lower case.

pwd → shows working directory (where you are)

ls → lists all the files and folders in the directory

cd → stands for 'change directory':

cd lab1 → change to the lab1 directory/folder

cd .. → going up one directory/folder

cd ../.. → going up two directories

Useful Unix Commands (Part 2)

mkdir lab1

make directory lab1 aka makes a folder called lab1

rm -r lab1

removes the directory lab1 (-r stands for recursive, which deletes any possible folders in lab1 that might contain other files)

cp lab1/file1.txt lab2

copies file1.txt file (inside of the folder lab1), to the folder lab2

mv lab1/file1.txt lab2 → moves a file called file1.txt, which is inside of the folder lab1, to the folder lab2

zip zipfile.zip file1.txt file2.txt file3.txt

zips files 1 to 3 into zipfile.zip

zip -r zipfile.zip lab1/

zips up all files in the lab1 folder into zipfile.zip

Useful Unix Commands (Part 3)

^c → ctrl + c, interrupts running program

^d →ctrl + d, gets you out of python3

"tab" autocompletes what you're typing based on the files in the current folder

"up" cycles through the commands you've typed. Similarly for the opposite effect press "down"

Useful Unix Commands (Part 4)

python3 -i test.py → load test.py in python3, and you can call the functions in test.py.

gedit lb1.txt & → opens up lb1.txt on gedit and & allows you to run your terminal at the same time (else your terminal pauses until you close gedit)

ssh -X ANDREW_ID@unix.andrew.cmu.edu → log into the Andrew servers and the files you've created from labs and the Linux cluster computers from your personal computer w/out setting anything up (replacing ANDREW_ID with your own andrewID)

And lastly, you can always do **man <command>** to find out more about a particular command you're interested about (eg. man cp, man ls)

Use the Resources page

- To **install Python 3** to your computers or
- To try out **remote access** instructions so that you can run Python on Andrew machines from your own machine
- To see other supporting resources.
- **If you don't have a computer**, learn the places and open-hours of labs that you can use. (Programming assignments should be submitted until 11:59 PM on its due date)

An Introduction to Programming - I -

Today

- Programing languages and programs
- The Python programming language

A programming “language” is a **form**al notation

Not a *natural* language

Recipe

White Wine and Cheddar Sauce
…make a roux, heat the mikl, add some of the warm milk…

- Interpreted by a person
- …for herself (“I want sauce”)
- Unclear? Can be figured out (What’s a “roux”? How much is “some?)
- Typos? Can be figured out (“mikl” means “milk”)

Computer program

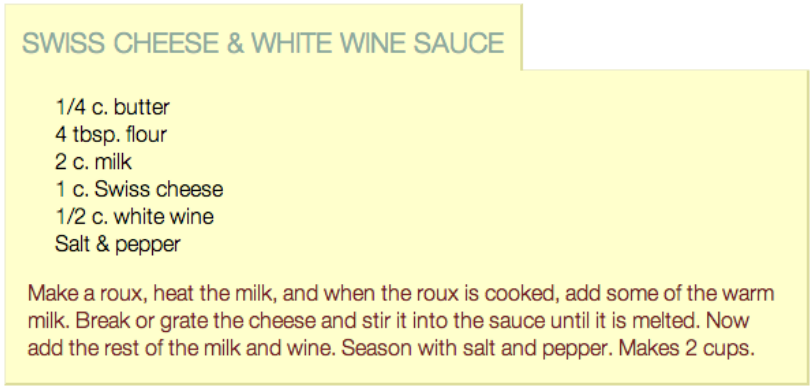
```
for i in range(5):
  print(whatever I want)
```

- Interpreted by a machine
- …for a human (“somebody wants to print something”)
- Unclear? Not a program (“whatever I want”????)
- Typos? Program errors (“pritn”???)

A programming “language” is a
formal notation
for **generalized** problem solving

Programs should be *general*

Recipe



SWISS CHEESE & WHITE WINE SAUCE

1/4 c. butter
4 tbsp. flour
2 c. milk
1 c. Swiss cheese
1/2 c. white wine
Salt & pepper

Make a roux, heat the milk, and when the roux is cooked, add some of the warm milk. Break or grate the cheese and stir it into the sauce until it is melted. Now add the rest of the milk and wine. Season with salt and pepper. Makes 2 cups.

Specific: "output" is two cups of sauce.

Program

```
def force(mass, accel) :
    return mass*accel
```

General: output is force for **any** combination of mass and acceleration.

Python

- Python is one of *many* programming languages.
- 2 widely used versions. We will use Python 3. (Specifically, Python version 3.3.2)

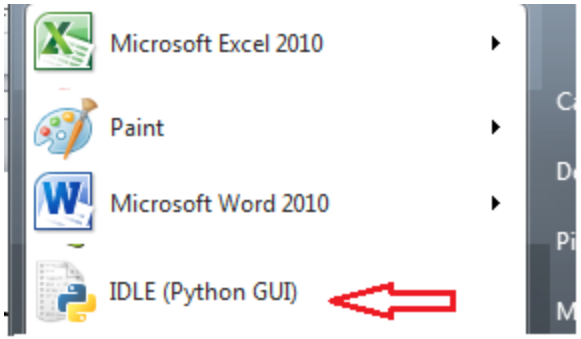
Running on the command line

> python3

or

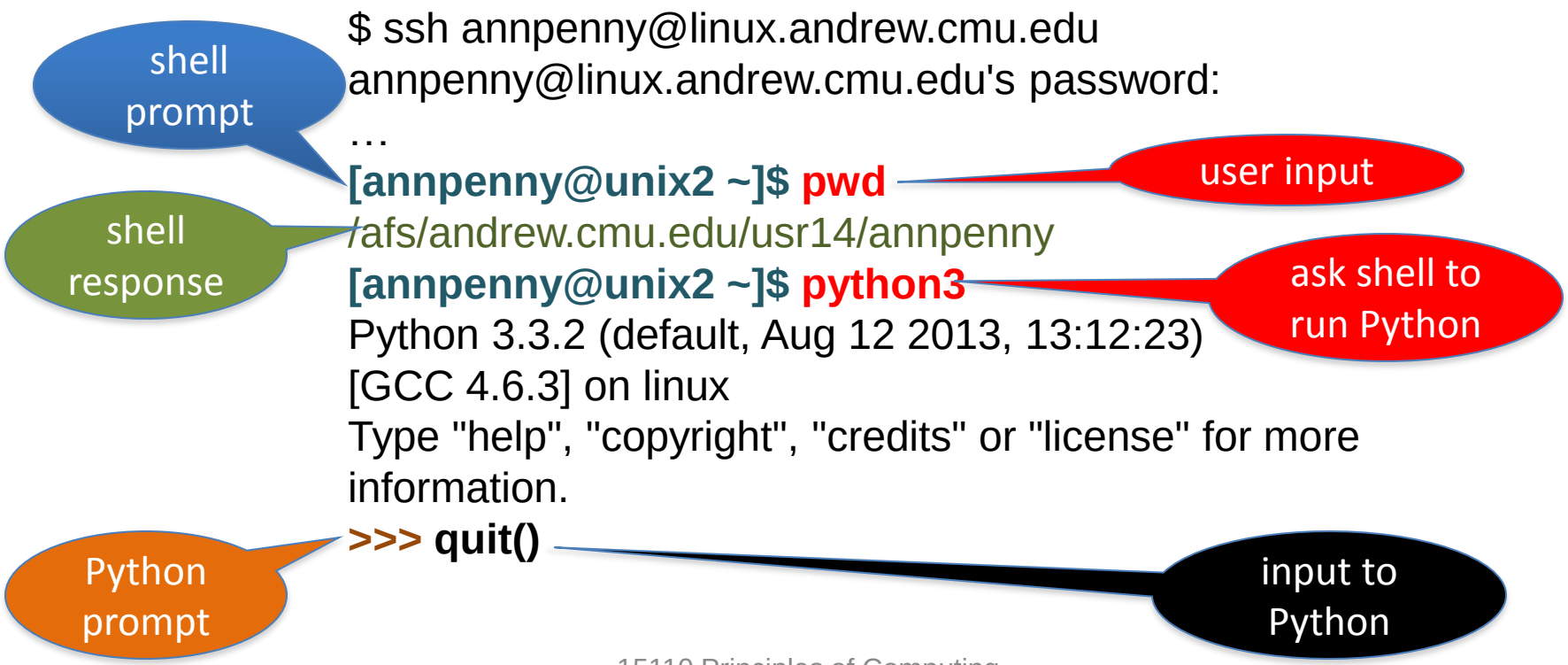
> python3 –i *filename.py*

Using IDLE



Command Line Interfaces

- Be aware of the difference between “talking to the shell” and “talking to Python”



Expressions and Statements

- *Know the difference!*

  Python **evaluates** an **expression** to get a *result* (number or other value)

  Python **executes** a **statement** to perform an action that has an *effect* (printing something, for example)

Arithmetic Expressions

- **Mathematical Operators**

+	Addition		
-	Subtraction	//	Integer division
*	Multiplication	**	Exponentiation
/	Division	%	Modulo (remainder)

- **Python is like a calculator:** type an expression and it tells you the value.

```
>> 2 + 3 * 5
⇒ 17
```

Order of Evaluation

Order of **operator precedence**:

Use **parentheses** to force alternate precedence

```
5*6+7 ≠ 5*(6+7)
```

Left associativity except for **

```
2 + 3 + 4    =(2+3)+4
2 ** 3 ** 4 = 2**(3**4)
```

Data Types

- **Integers**

```
4        15110-53        0
```

- **Floating Point Numbers**

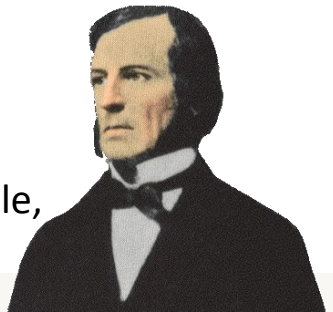
```
4.0 -0.8  0.33333333333333
7.34e+014
```

- **Strings**

```
"hello"   "A"    " "     ""        "7up!"
'there'      '"'    '15110'
```

- **Booleans**

```
True     False
```



George Boole,
1815-1864

Integer Division

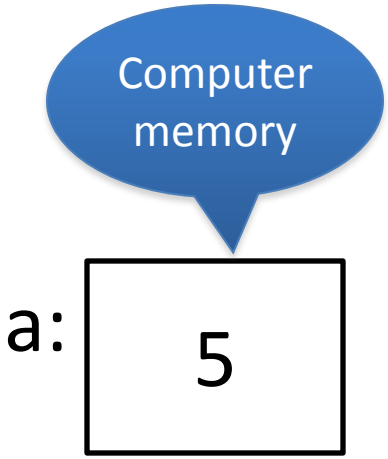
In Python3:

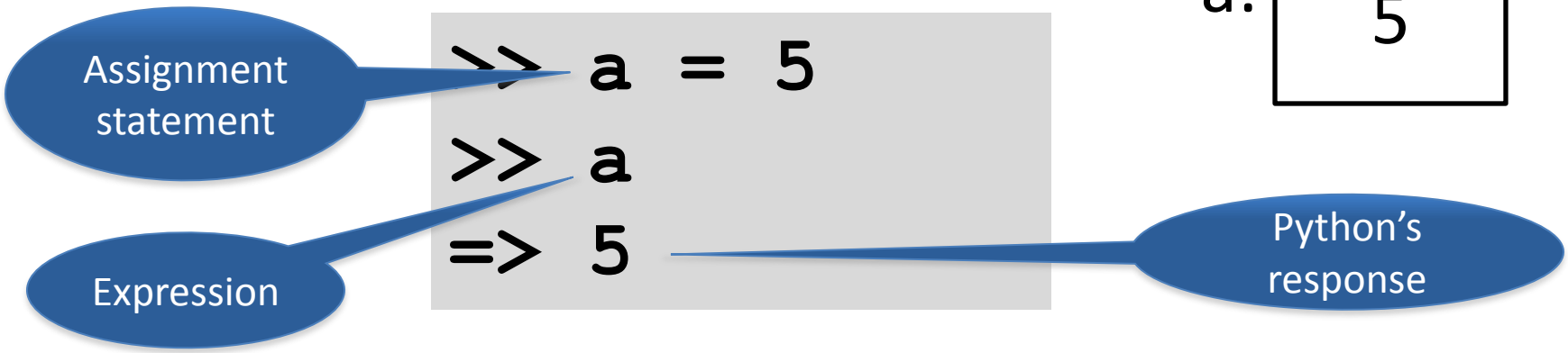
7 / 2	equals	**3.5**
7 // 2	equals	**3**
7 // 2.0	equals	**3.0**
7.0 // 2	equals	**3.0**
-7 // 2	equals	**-4**

// opertor **rounds down to smaller number,** not towards zero

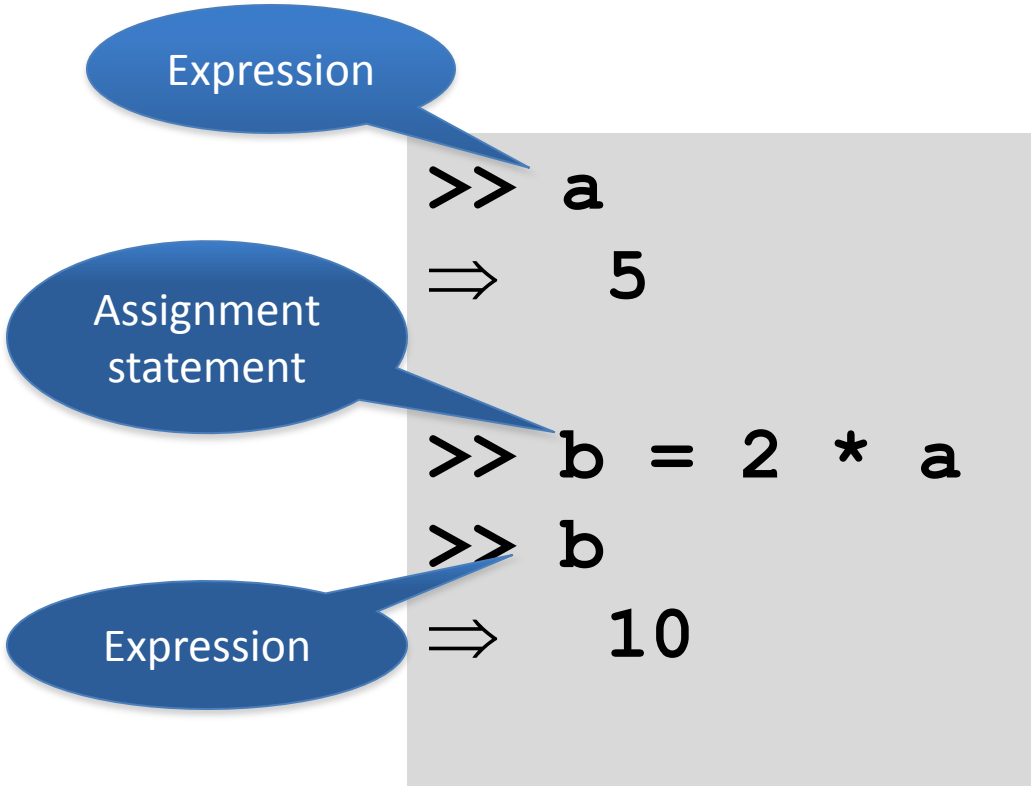
Variables

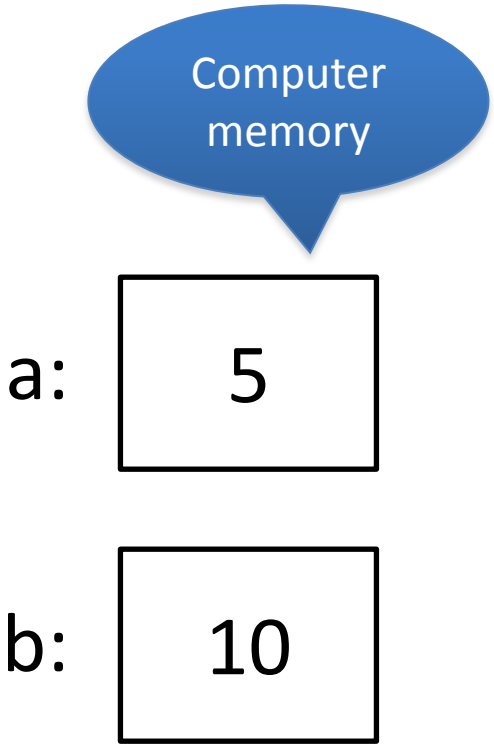
- A variable is *not* an "unknown" as in algebra.
- In computer programming, a variable is a *place* where you can store a value.
- In Python we store a value using an *assignment statement:*





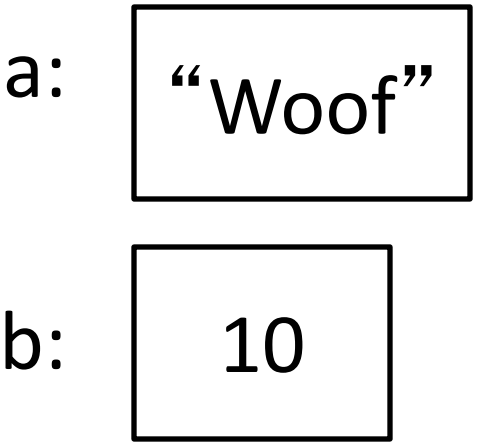
Variables


Expression
Assignment statement
Expression
>> a
⇒ 5
>> b = 2 * a
>> b
⇒ 10


Computer memory
a:
5
b:
10

Variables

```
>> a
⇒  5
>> b
⇒  10
>> a = “Woof”
>> a
⇒  “Woof”
>> b
⇒  10
```



Variable b does not “remember” that its value came from variable a.

Variable Names

- All variable names must **start with a letter** (lowercase recommended).

- The remainder of the variable name can consist of any combination of uppercase or lowercase **letters, digits and underscores** (_).

- Identifiers in Python are **case sensitive**. Example: `Value` is not the same as `value`.

Built-In Functions (Methods)

- Lots of math stuff, e.g., sqrt, log, sin, cos

```
import math
r = 5 + math.sqrt(2)
alpha = math.sin(math.pi/3)
```

Using predefined modules

- `math` is a predefined module of **functions** (also called **methods**) that we can use without writing their **implementations**.

```
math.sqrt(16)
math.pi
math.sin(math.pi / 2)
```

Write Your Own Methods

```
def tip (total):
    return total * 0.18
```

```
>>  tip(100)
⇒   18.0
>>  tip(135.72)
⇒   24.4296
```

Method Syntax

`def` *methodname*(*parameterlist*)**:**
□□□□*instructions*

- **`def`** is a reserved word and cannot be used as a variable name.

- **Indentation** *is critical.*
  Use spaces *only,* **not** tabs **!!!!!!!!!!!!!!!!!!!!!!!!**

Methods are **general**

- The **parameter list** can contain 1 or more variables that represent data to be used in the method's computation.

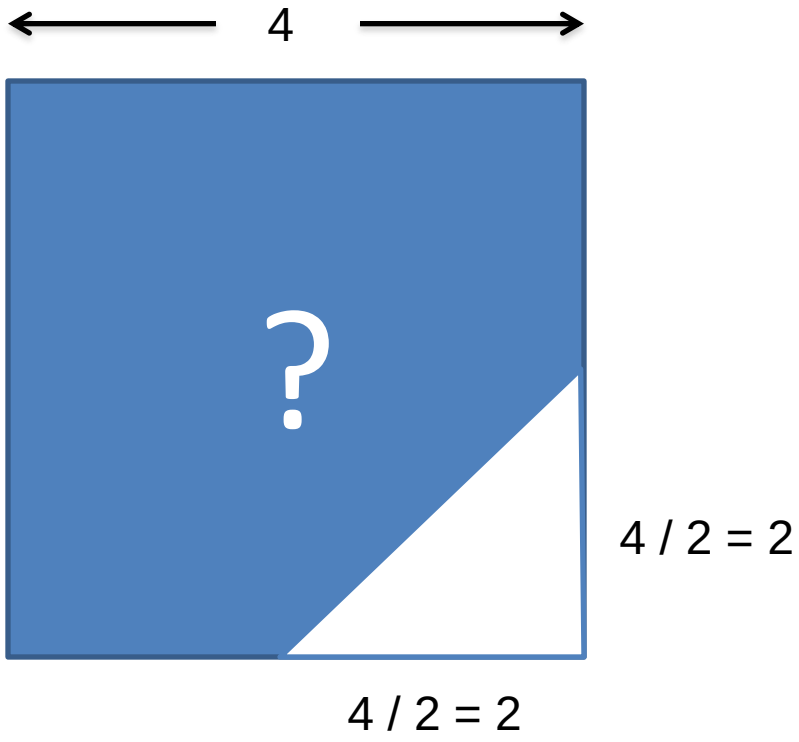
- A method can also have **no parameters**!

```
def hello_world():
    print("Hello World!\n")
```

parentheses must be present!

(`\n` is a newline character)

Example: area of a countertop



countertop.py

```
def compute_area():
  square = 4 * 4
  triangle = 0.5 * (4/2) * (4/2)
  area = square - triangle
  return area
```

empty parameter list

Calling the function/method :

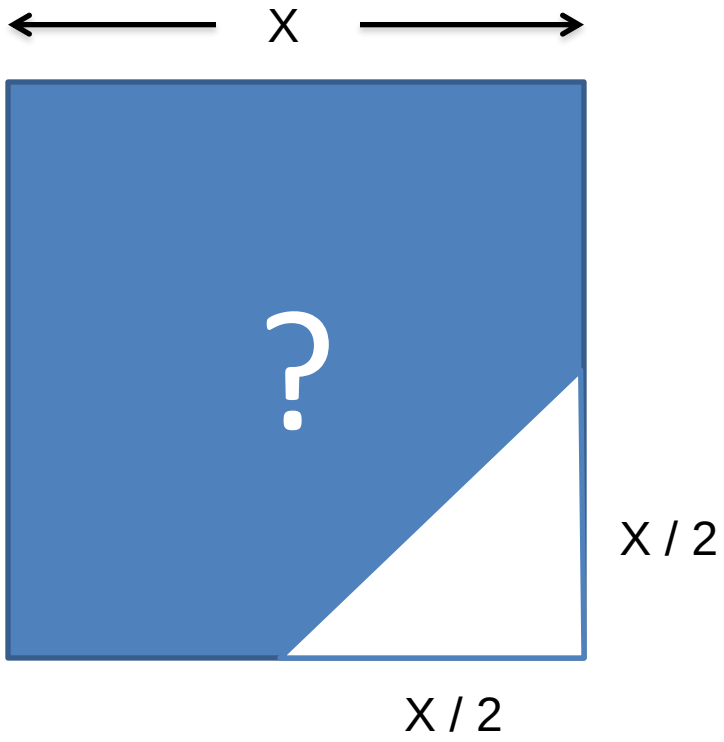
```
> python3 -i countertop.py
```

(OR run from IDLE)

```
>>> compute_area()
14.0
```

empty argument list

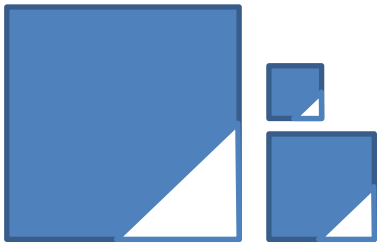
Generalizing the problem



countertop.py

```
def compute_area(side):
    square = side * side
    triangle = 0.5 * (side/2 * side/2)
    area = square - triangle
    return area
```

parameter



To run (use) the function/method:

```
python3 -i countertop.py
>>> compute_area(109)
```

(OR run from IDLE)

argument

(run function with side = 109)

Method inputs are for **generality**

- `def compute_area(side):`
  `…`
  `side` names the input parameter to the method

parameter

- `>>> compute_area(109)` ← argument

(run function with side = 109)

`109` is the argument value to substitute for the parameter `side`

- But we can use **any positive number** and get an answer that makes sense!

Method Outputs

Method outputs a value by return

```
def tip (total):
    return total * 0.18
```

… or it may return None

```
def hello_world():
    print("Hello World!\n”)
```

Method Outputs

```
>>> tip(12)
2.16
```

value returned

```
>>> print(tip(12))
2.16
```

value returned and printed

value printed by method

```
>>> hello_world()
Hello World!
```

value returned and printed

```
>>> print(hello_world())
Hello World!
None
```

Method Outputs

- To use a method, we "**call**" the method.
- A method can **return either one answer** or **no answer** (`None`) to its "caller".
- The `hello_world` function **does not return** anything to its caller. It simply prints something on the screen.
- The `tip` function **does return** its result to its caller so it can use the value in another computation: `tip(12) + tip(16)`

Method Outputs

Suppose we write `compute_area` this way:

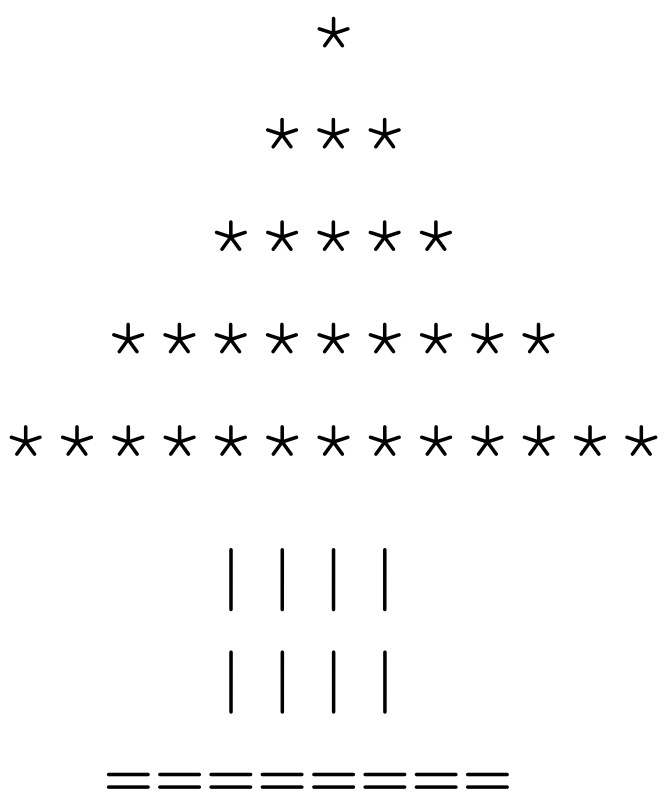
```
def compute_area(side):
    square = side * side
    triangle = 0.5 * side/2 * side/2
    area = square - triangle
    print(area)
```

Now the following computation does not work since each function call prints but returns `None`:

```
compute_area(109) + compute_area(78)
```

Which methods would you write

for a program which draws



escape.py

(a function with two parameters)

```
import math
def compute_ev(mass, radius):
  # computes escape velocity
  univ_grav = 6.67e-011
  return math.sqrt(2*univ_grav*mass/radius)
```

Comments begin with #

To run the function for Earth in `python3`:

```
python3 -i escape.py
>>> compute_ev(5.9742e+024, 6378.1)
```

What Could Possibly Go Wrong?

```
alpha = 5
2 + alhpa

3 / 0
import math
math.sqrt(-1)
math.sqrt(2, 3)
```

Next Lecture

Loops – how to run a million computations with only a few lines of code.