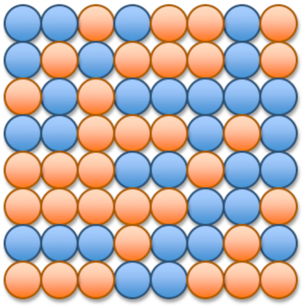


UNIT 2A
An Introduction to Programming

Python

- Python is one of *many* programming languages.
- 2 widely used versions. We will use Python 3.
- Running Python 3 on the command line:

```
> python3 –i filename.py
```

(–i means interactive mode)

  - python3 is an interpreter that translates python instructions (that we can read) to binary/machine language that the computer understands.

Arithmetic Expressions

- Mathematical Operators

+	Addition	/	Division
-	Subtraction	%	Modulo
*	Multiplication	**	Exponentiation

- Order of Precedence
  {**}   then {* / %}   then {+ -}
- 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     0.3333333333333333
7.34e+014
```

- Strings

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

- Booleans

```
True   False
```

George Boole,
1815-1864

Integer Division

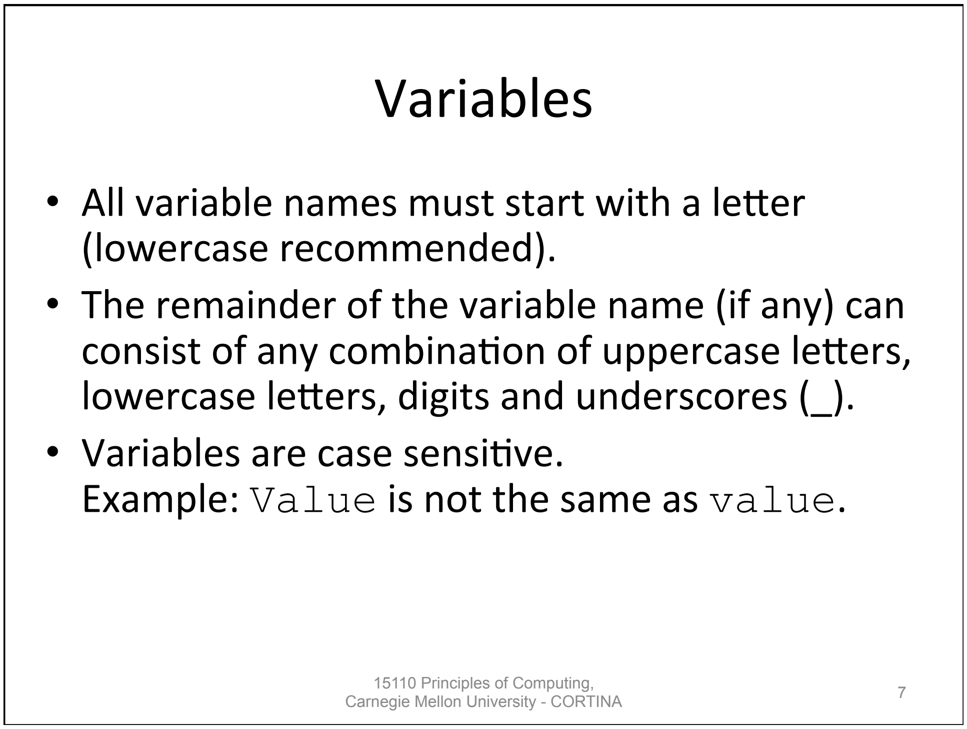
In Python3:

- 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** (beware! // rounds **down**)

Modulo

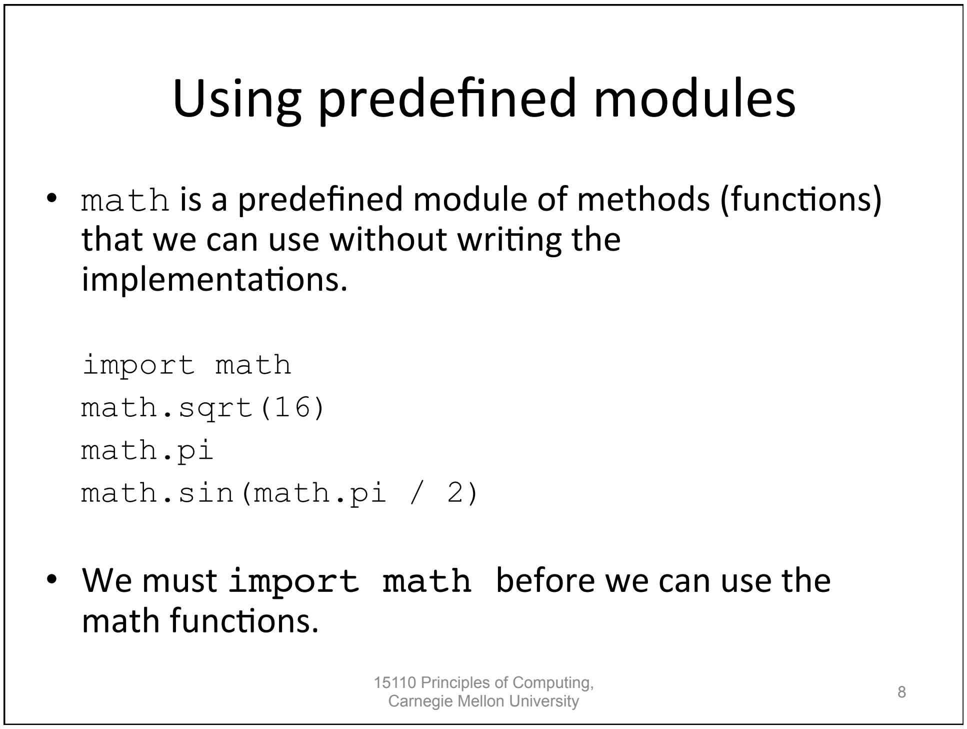
In Python3:

- 7 % 2 equals **1**
- 15 % 4 equals **3**
- 42 % 7 equals **0**
- 6 % 14 equals **6**
- -7 % 2 equals **1** (think about it…)



Variables

- All variable names must start with a letter (lowercase recommended).
- The remainder of the variable name (if any) can consist of any combination of uppercase letters, lowercase letters, digits and underscores (_).
- Variables are case sensitive.
  Example: `Value` is not the same as `value`.



Using predefined modules

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

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

- We must **`import math`** before we can use the math functions.

Assignment Statements

- The lefthand side must contain a single variable.
- The righthand side can be any valid Python expression:
  - A numerical, string or boolean value.
    `x = 45.2`
  - A numerical expression.
    `y = x * 15`
  - A method (function) call.
    `z = math.sqrt(15110)`
  - Any combination of these:
    `root1 = -b + math.sqrt(b*b-4*a*c)/(2*a)`

Methods

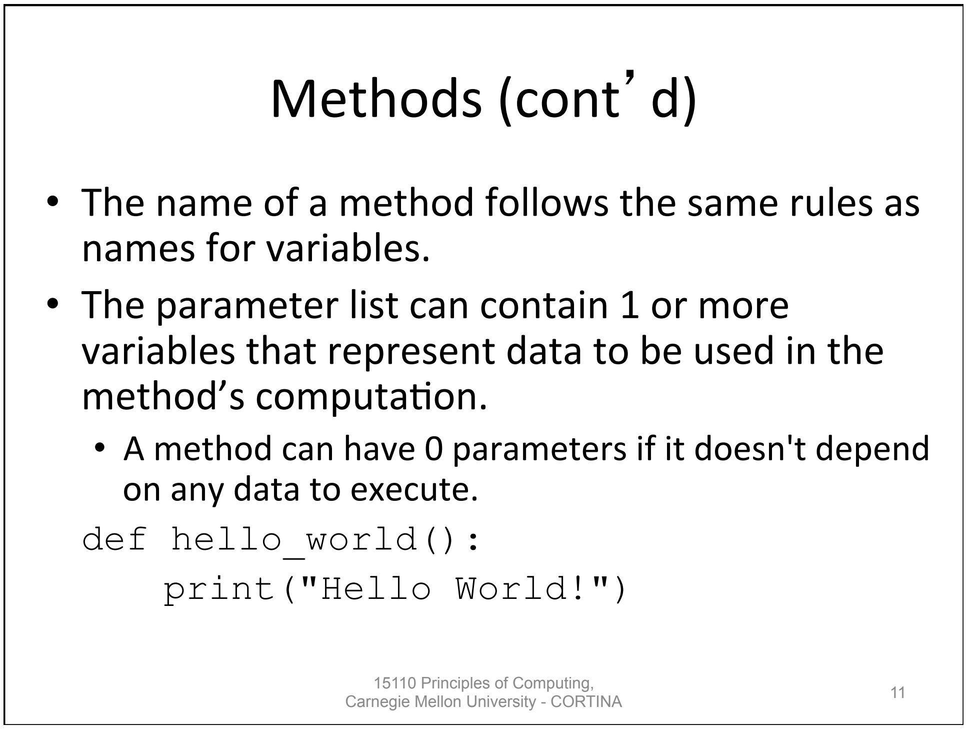
- Methods are used to capture small algorithms that might be repeated with different initial conditions.

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

  □□□□*instruction1*

  □□□□*instruction2*

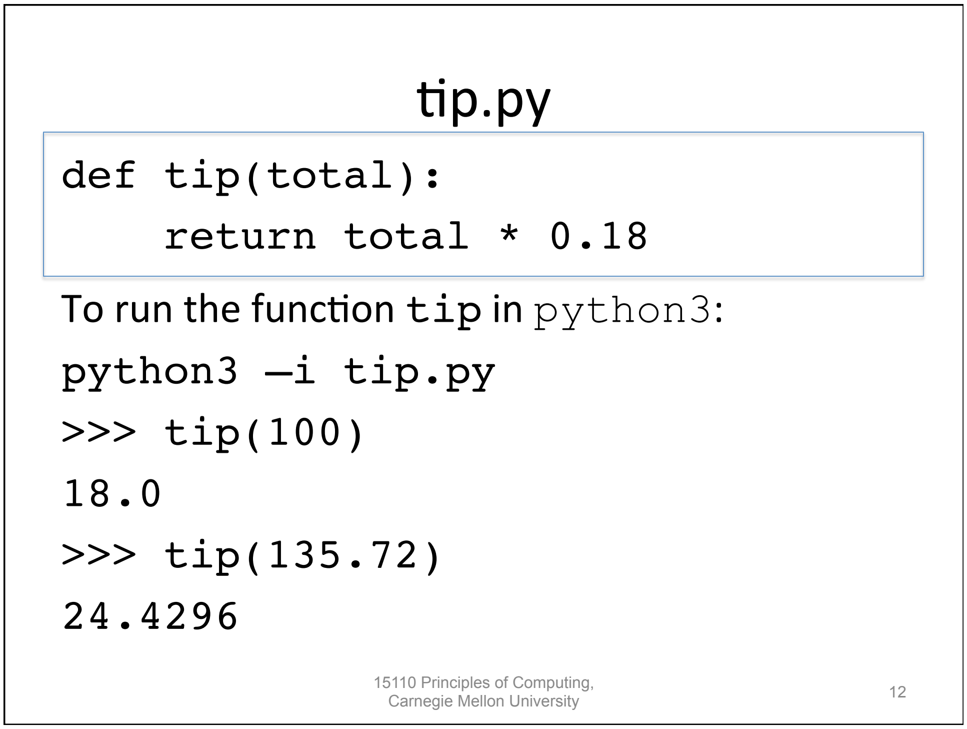
  etc.
- `def` is a reserved word and cannot be used as a variable name.
- *Indentation is critical.* Use spaces *only.*



Methods (cont’ d)

- The name of a method follows the same rules as names for variables.
- The parameter list can contain 1 or more variables that represent data to be used in the method’s computation.
  - A method can have 0 parameters if it doesn't depend on any data to execute.

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

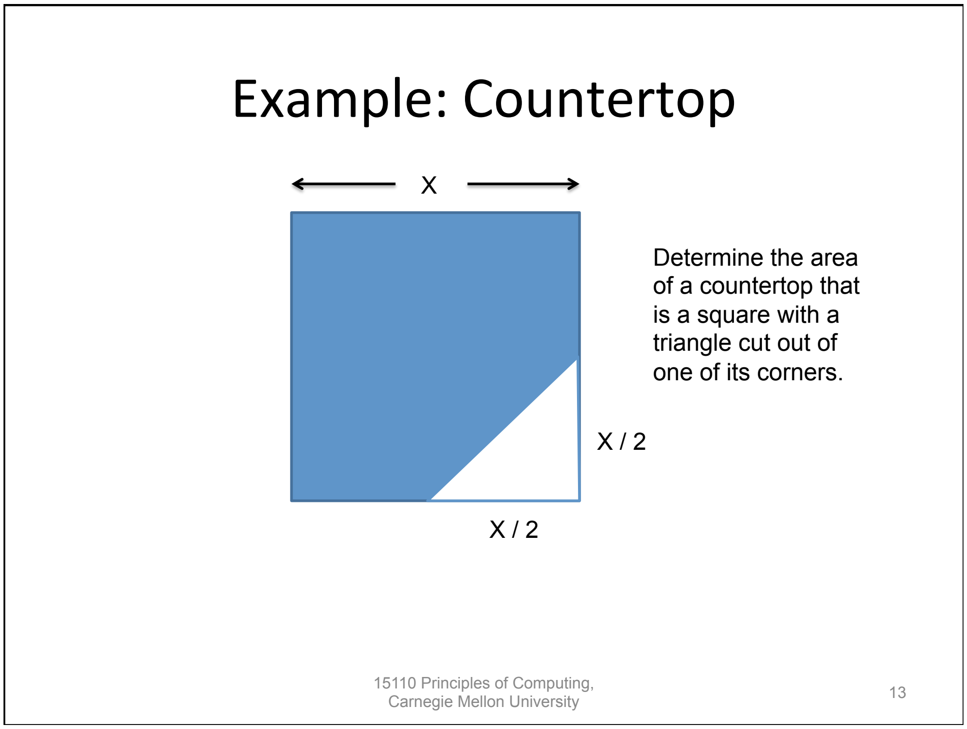


tip.py

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

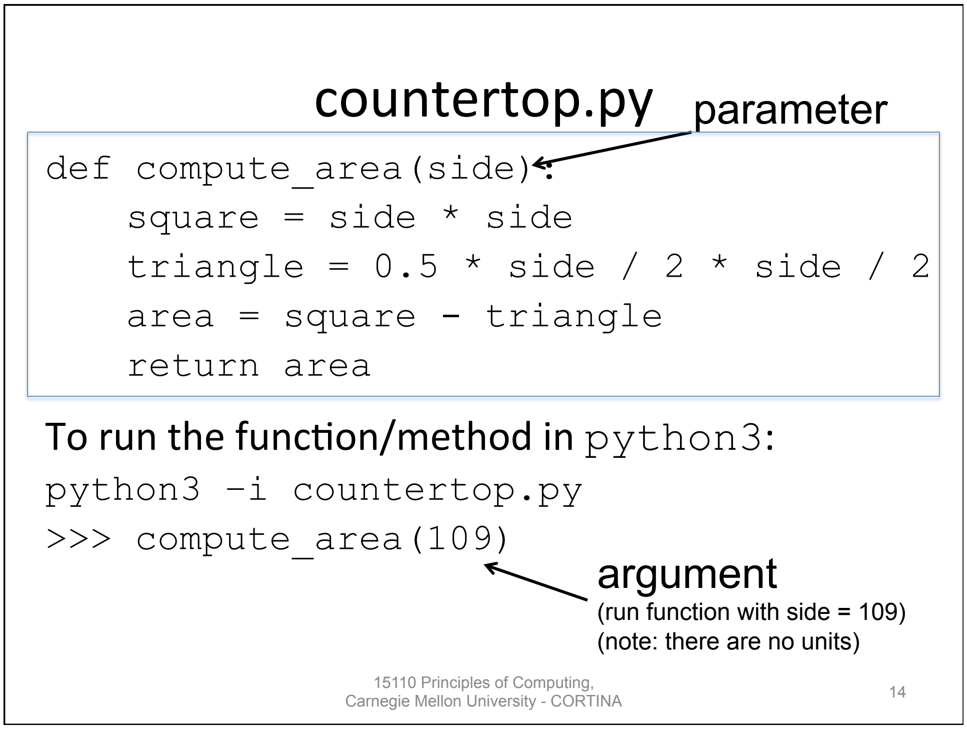
To run the function `tip` in `python3`:

```
python3 –i tip.py
>>> tip(100)
18.0
>>> tip(135.72)
24.4296
```



Example: Countertop

Determine the area of a countertop that is a square with a triangle cut out of one of its corners.



countertop.py

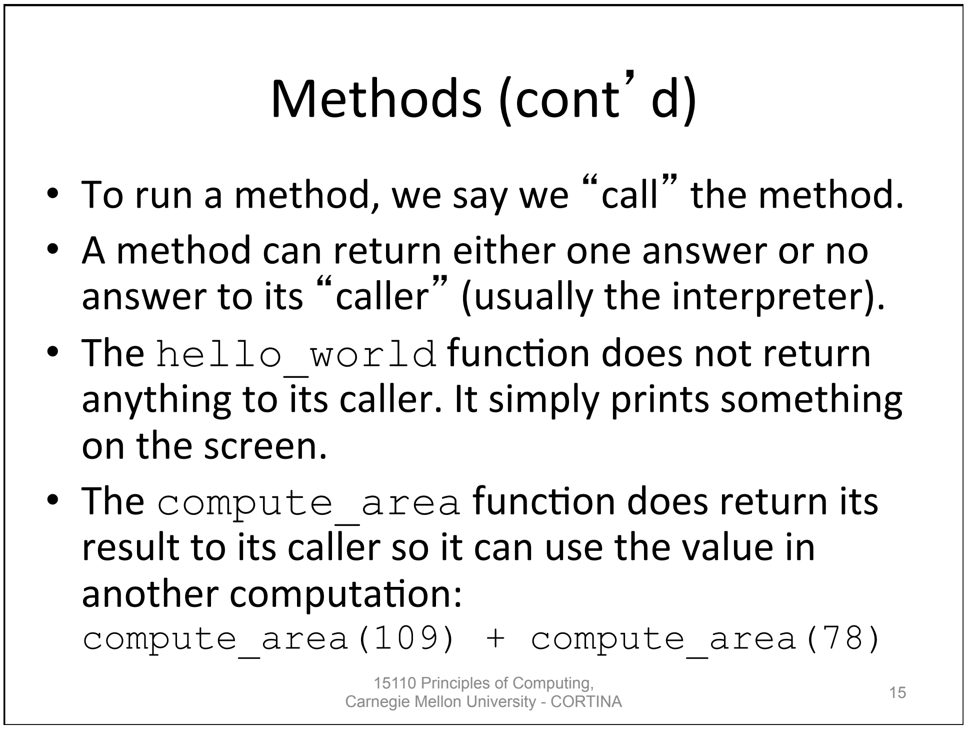
parameter

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

To run the function/method in `python3`:

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

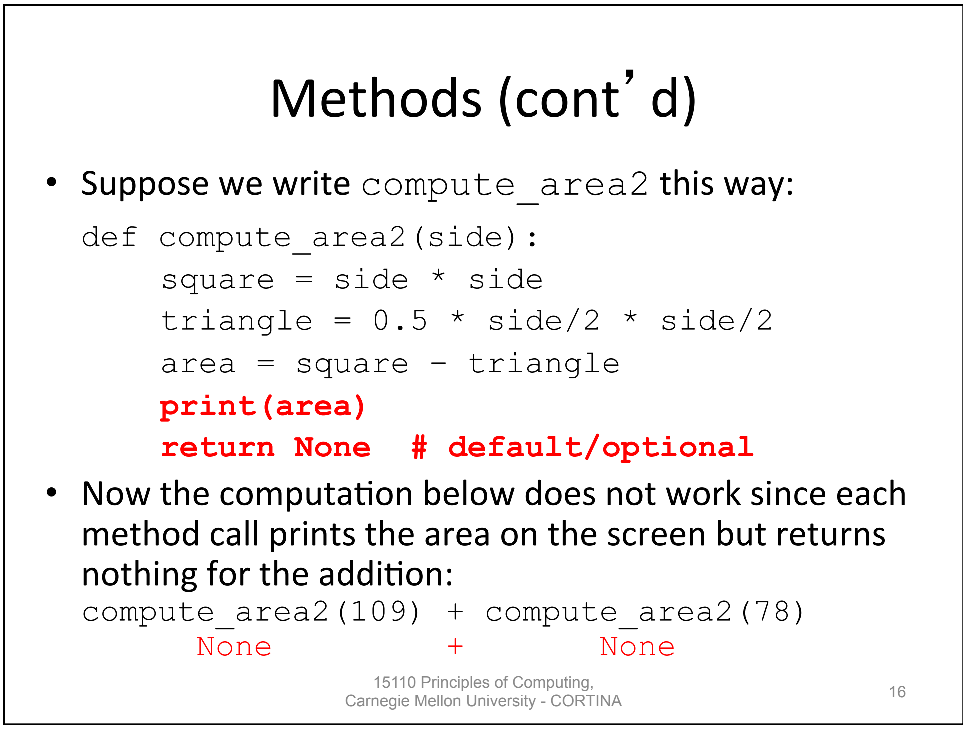
argument
(run function with side = 109)
(note: there are no units)



Methods (cont’d)

- To run a method, we say we “call” the method.
- A method can return either one answer or no answer to its “caller” (usually the interpreter).
- The `hello_world` function does not return anything to its caller. It simply prints something on the screen.
- The `compute_area` function does return its result to its caller so it can use the value in another computation:

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



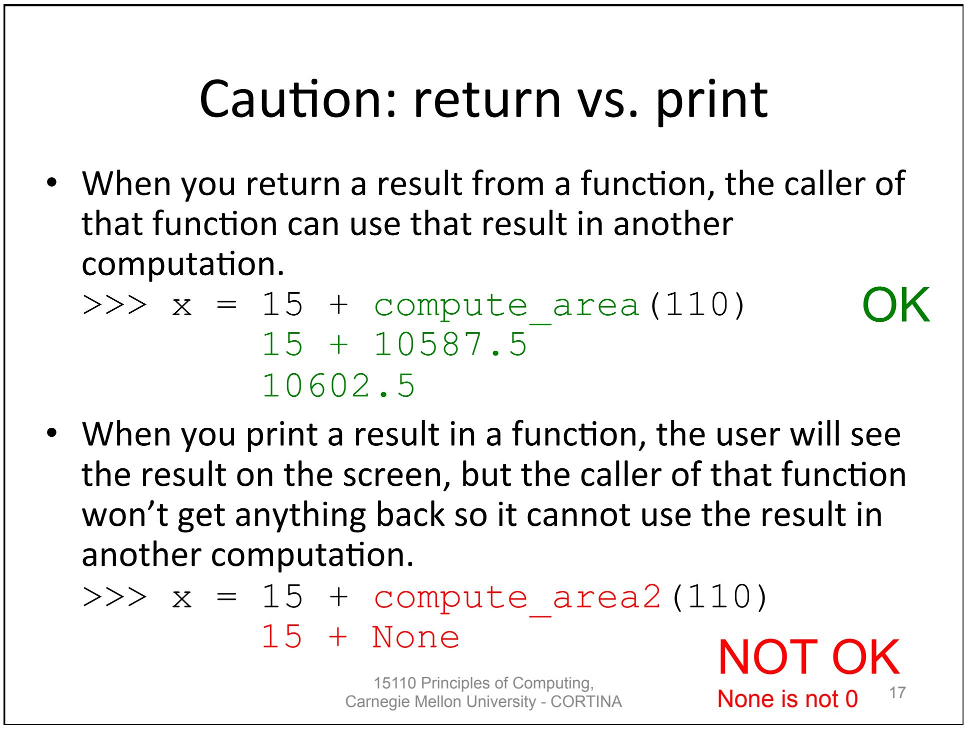
Methods (cont’d)

- Suppose we write `compute_area2` this way:

```
def compute_area2(side):
    square = side * side
    triangle = 0.5 * side/2 * side/2
    area = square - triangle
    print(area)
    return None  # default/optional
```

- Now the computation below does not work since each method call prints the area on the screen but returns nothing for the addition:

```
compute_area2(109) + compute_area2(78)
      None           +       None
```



Caution: return vs. print

- When you return a result from a function, the caller of that function can use that result in another computation.

```
>>> x = 15 + compute_area(110)
            15 + 10587.5
            10602.5
```

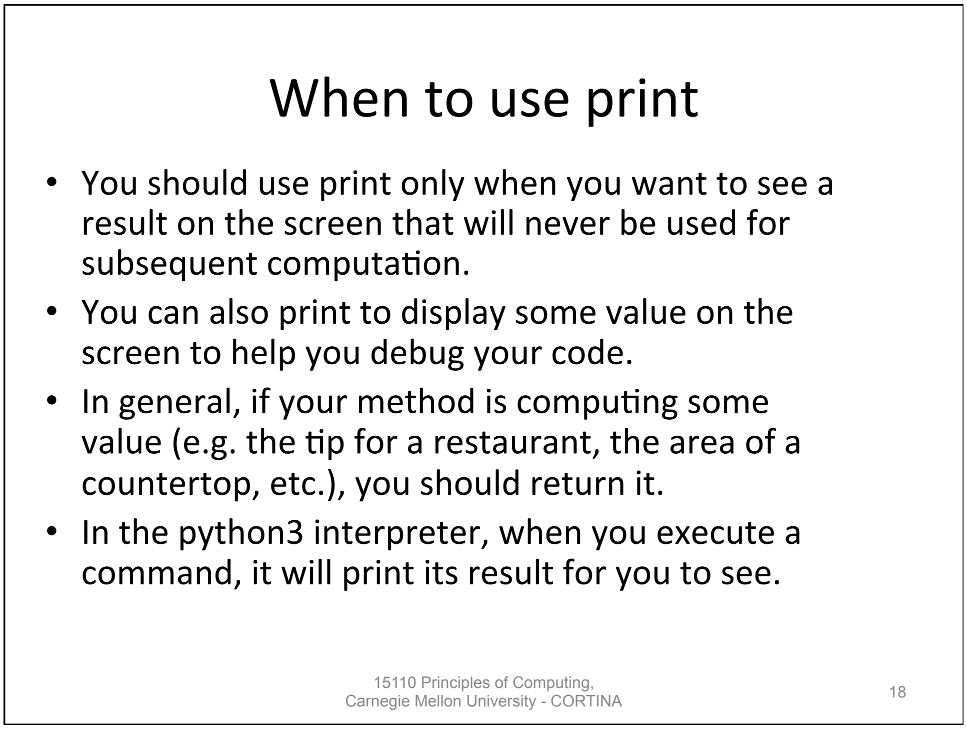
OK

- When you print a result in a function, the user will see the result on the screen, but the caller of that function won’t get anything back so it cannot use the result in another computation.

```
>>> x = 15 + compute_area2(110)
            15 + None
```

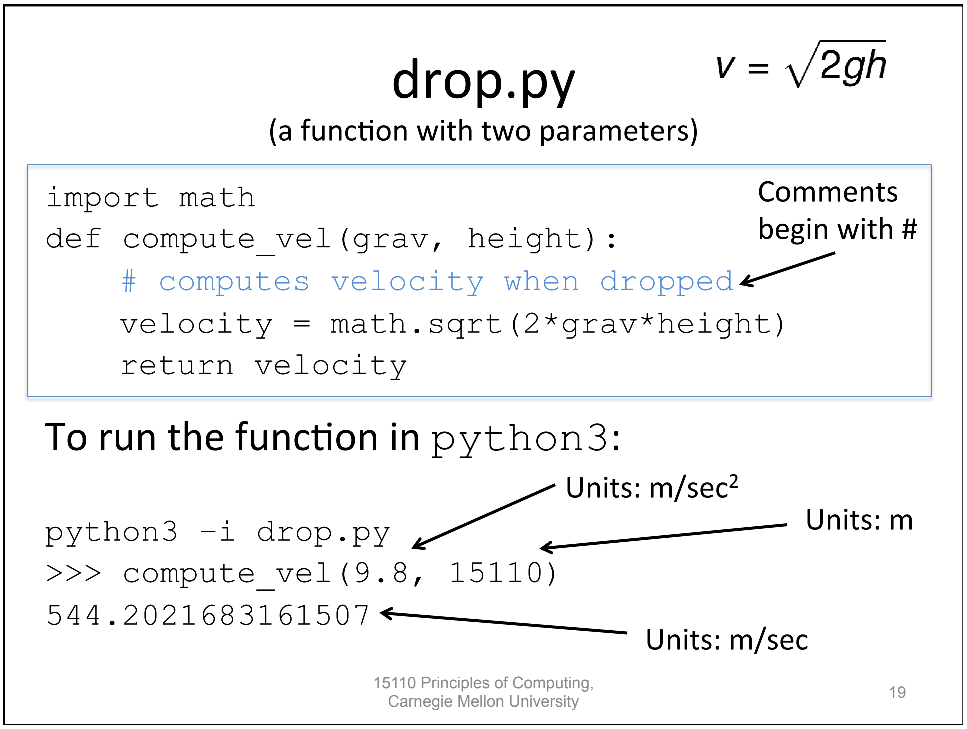
NOT OK

None is not 0



When to use print

- You should use print only when you want to see a result on the screen that will never be used for subsequent computation.
- You can also print to display some value on the screen to help you debug your code.
- In general, if your method is computing some value (e.g. the tip for a restaurant, the area of a countertop, etc.), you should return it.
- In the python3 interpreter, when you execute a command, it will print its result for you to see.



drop.py

$v = \sqrt{2gh}$

(a function with two parameters)

```
import math
def compute_vel(grav, height):
    # computes velocity when dropped
    velocity = math.sqrt(2*grav*height)
    return velocity
```

Comments begin with #

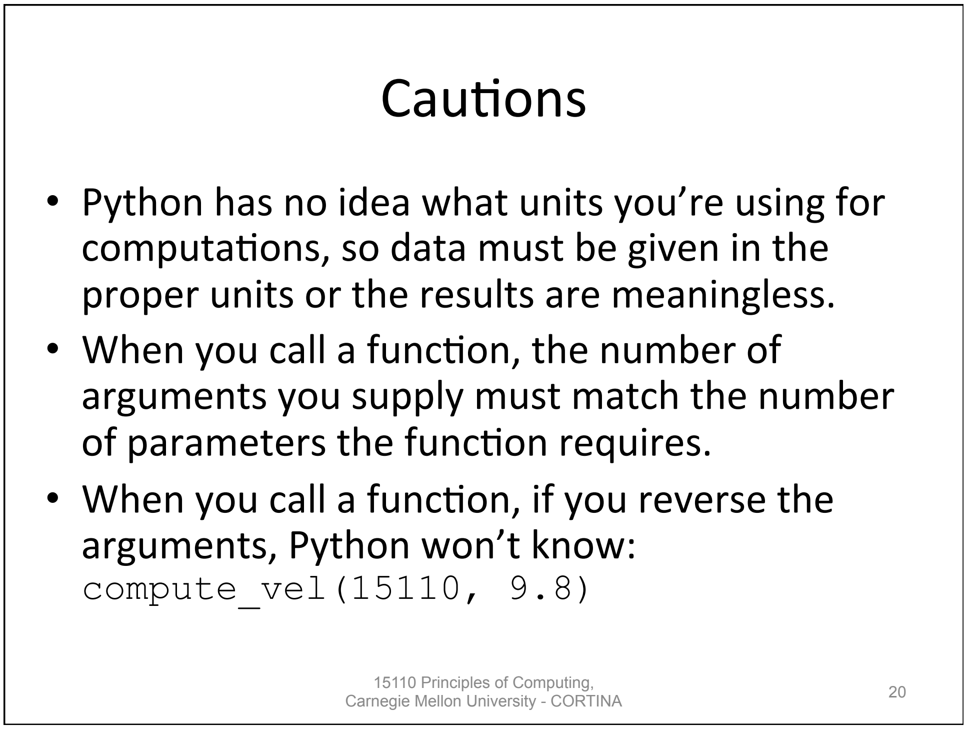
To run the function in `python3`:

```
python3 –i drop.py
>>> compute_vel(9.8, 15110)
544.2021683161507
```

Units: m/sec^2

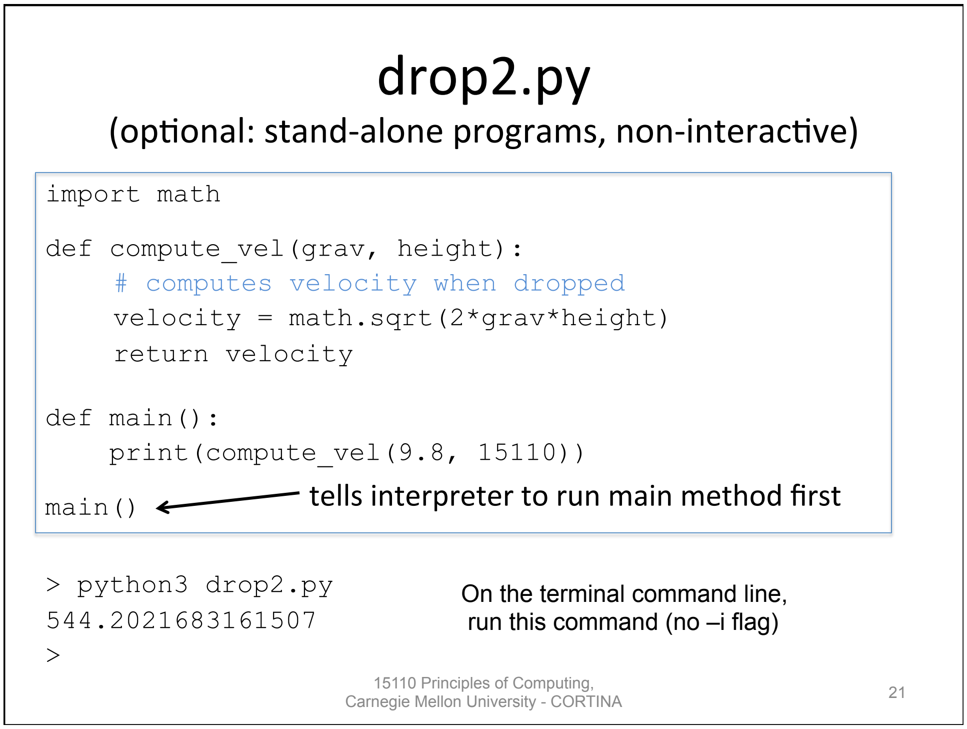
Units: m

Units: m/sec



Cautions

- Python has no idea what units you're using for computations, so data must be given in the proper units or the results are meaningless.
- When you call a function, the number of arguments you supply must match the number of parameters the function requires.
- When you call a function, if you reverse the arguments, Python won't know:
  `compute_vel(15110, 9.8)`



drop2.py

(optional: stand-alone programs, non-interactive)

```
import math

def compute_vel(grav, height):
     # computes velocity when dropped
     velocity = math.sqrt(2*grav*height)
     return velocity

def main():
    print(compute_vel(9.8, 15110))

main()
```

tells interpreter to run main method first

```
> python3 drop2.py
544.2021683161507
>
```

On the terminal command line,
run this command (no –i flag)