Selected Programming Language: Introduction to Python

CS 427

Faculty of Science, Department of Mathematics

Ref:

Richard L. Halterman, LEARNING TO PROGRAM WITH PYTHON, 2011

http://python.cs.southern.edu/pythonbook/pythonbook.pdf

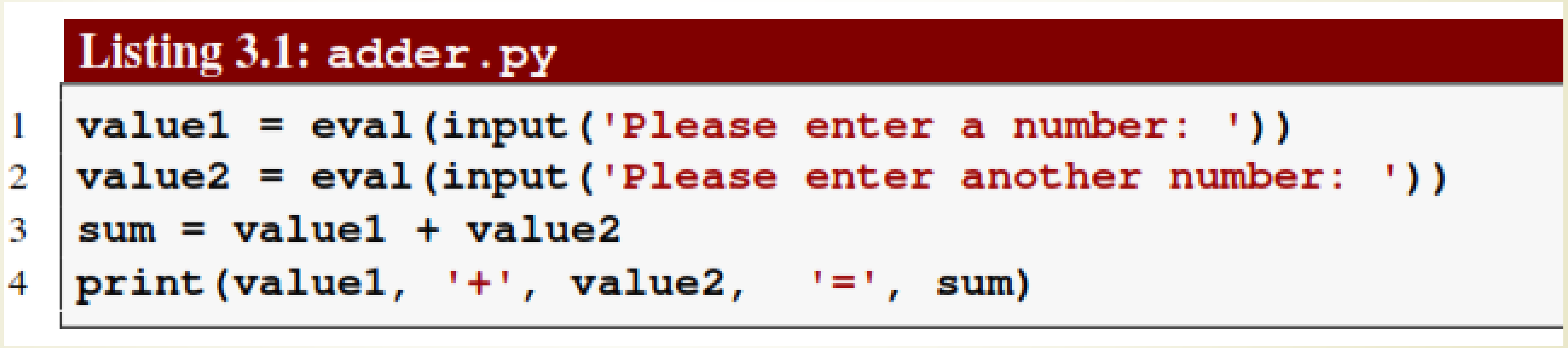
Chapter 3

Expressions and Arithmetic

Contents

Expressions

- A literal value like 34 and a variable like x are examples of a simple expressions.
- Values and variables can be combined with operators to form more complex expressions.

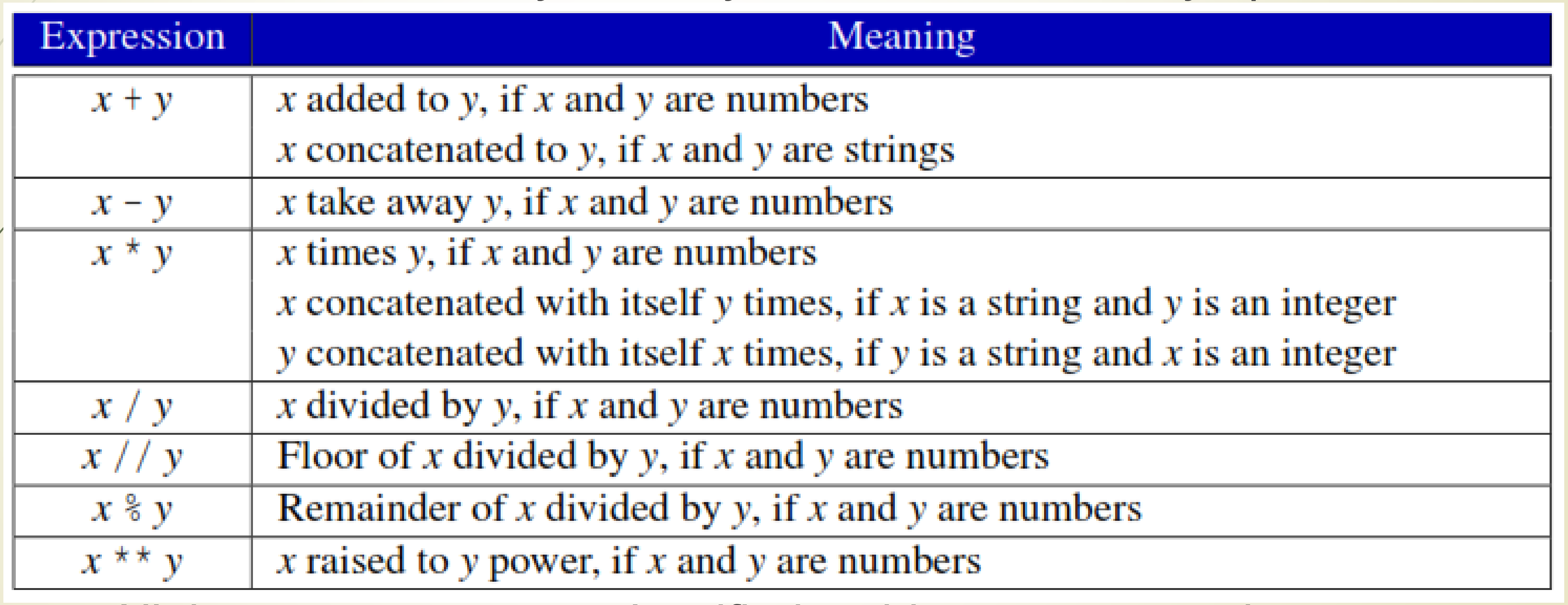


Listing 3.1: adder.py

```
value1 = eval(input('Please enter a number: '))
value2 = eval(input('Please enter another number: '))
sum = value1 + value2
print(value1, '+', value2,  '=', sum)
```

Expressions

Table 3.1: Commonly used Python arithmetic binary operators

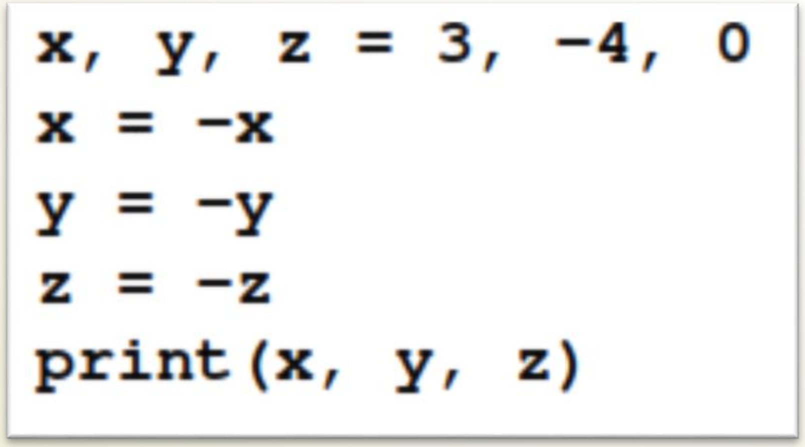


Expression	Meaning
x + y	x added to y, if x and y are numbers x concatenated to y, if x and y are strings
x - y	x take away y, if x and y are numbers
x * y	x times y, if x and y are numbers x concatenated with itself y times, if x is a string and y is an integer y concatenated with itself x times, if y is a string and x is an integer
x / y	x divided by y, if x and y are numbers
x // y	Floor of x divided by y, if x and y are numbers
x % y	Remainder of x divided by y, if x and y are numbers
x ** y	x raised to y power, if x and y are numbers

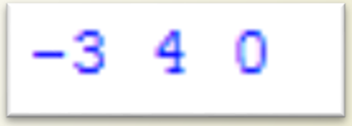
All these operators are classified as binary operators because they operate on two operands

Expressions

- Two operators, + and -, can be used as unary operators. A unary operator has only one operand.
- The unary operator expects a single numeric expression (literal number, variable, or more complicated numeric expression within parentheses) immediately to its right; it computes the *additive inverse* of its operand.



```
x, y, z = 3, -4, 0
x = -x
y = -y
z = -z
print(x, y, z)
```



```
-3 4 0
```

Expressions

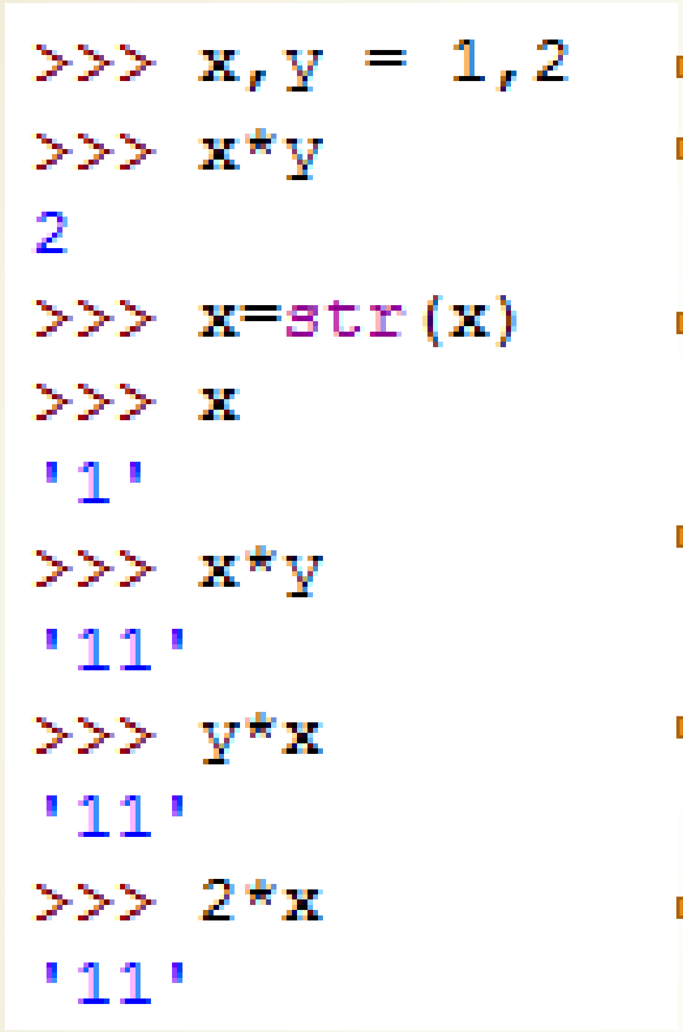
- The following statement

  print(-(4 - 5))

  within a program would print 1
- The unary + operator is present only for completeness; when applied to a numeric value, variable, or expression, the resulting value is no different from the original value of its operand.
- Omitting the unary + operator from the following statement

  ```
  x = +y
  ```

  does not change its behavior.



```
>>> x,y = 1,2
>>> x*y
2
>>> x=str(x)
>>> x
'1'
>>> x*y
'11'
>>> y*x
'11'
>>> 2*x
'11'
```

Tuple assignment of 2 integers

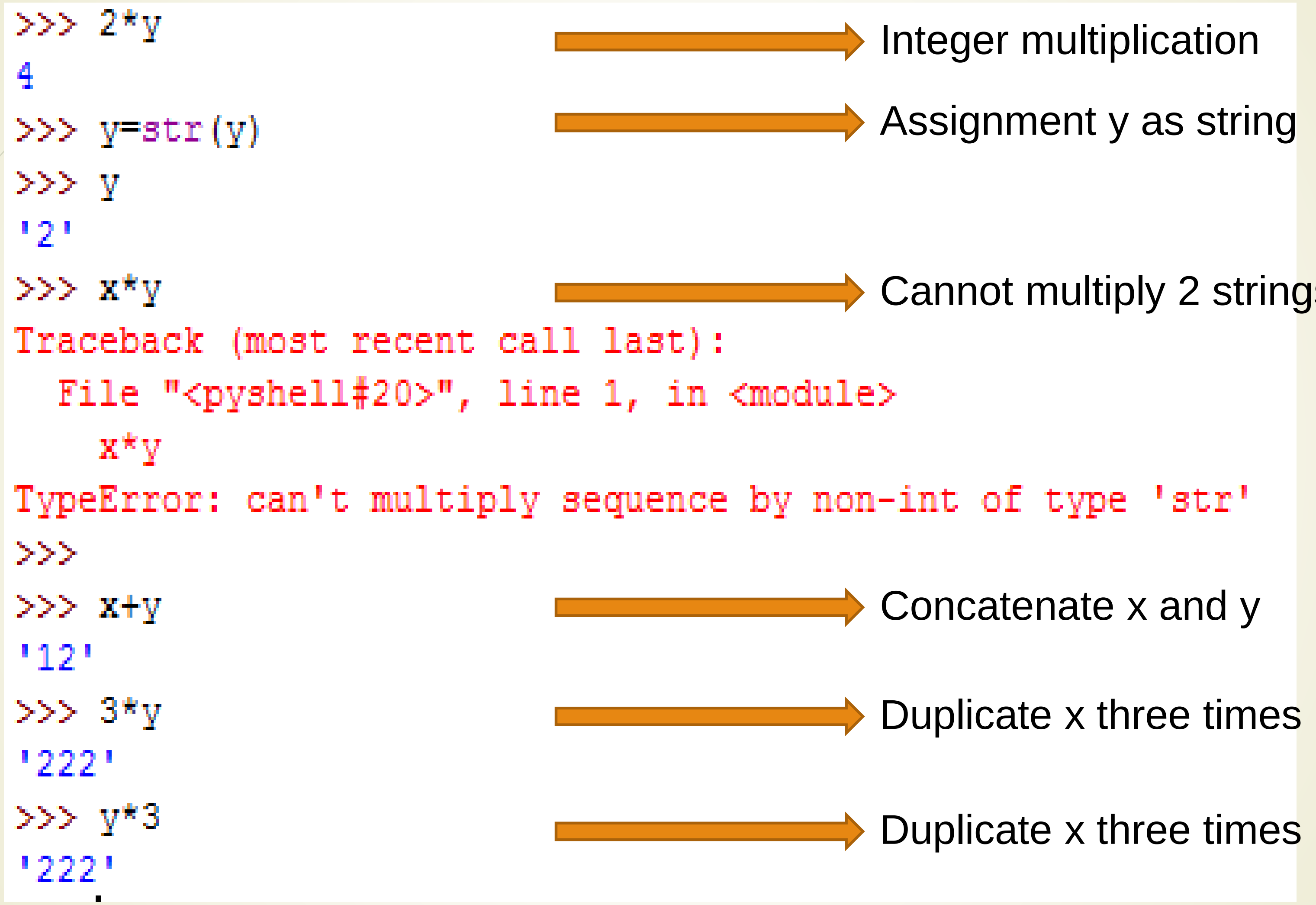
Integer multiplication

Assignment x as string

Duplicate x by y times

Duplicate x by y times

Duplicate x twice



```
>>> 2*y
4
>>> y=str(y)
>>> y
'2'
>>> x*y
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    x*y
TypeError: can't multiply sequence by non-int of type 'str'
>>>
>>> x+y
'12'
>>> 3*y
'222'
>>> y*3
'222'
```

Integer multiplication

Assignment y as string

Cannot multiply 2 strings

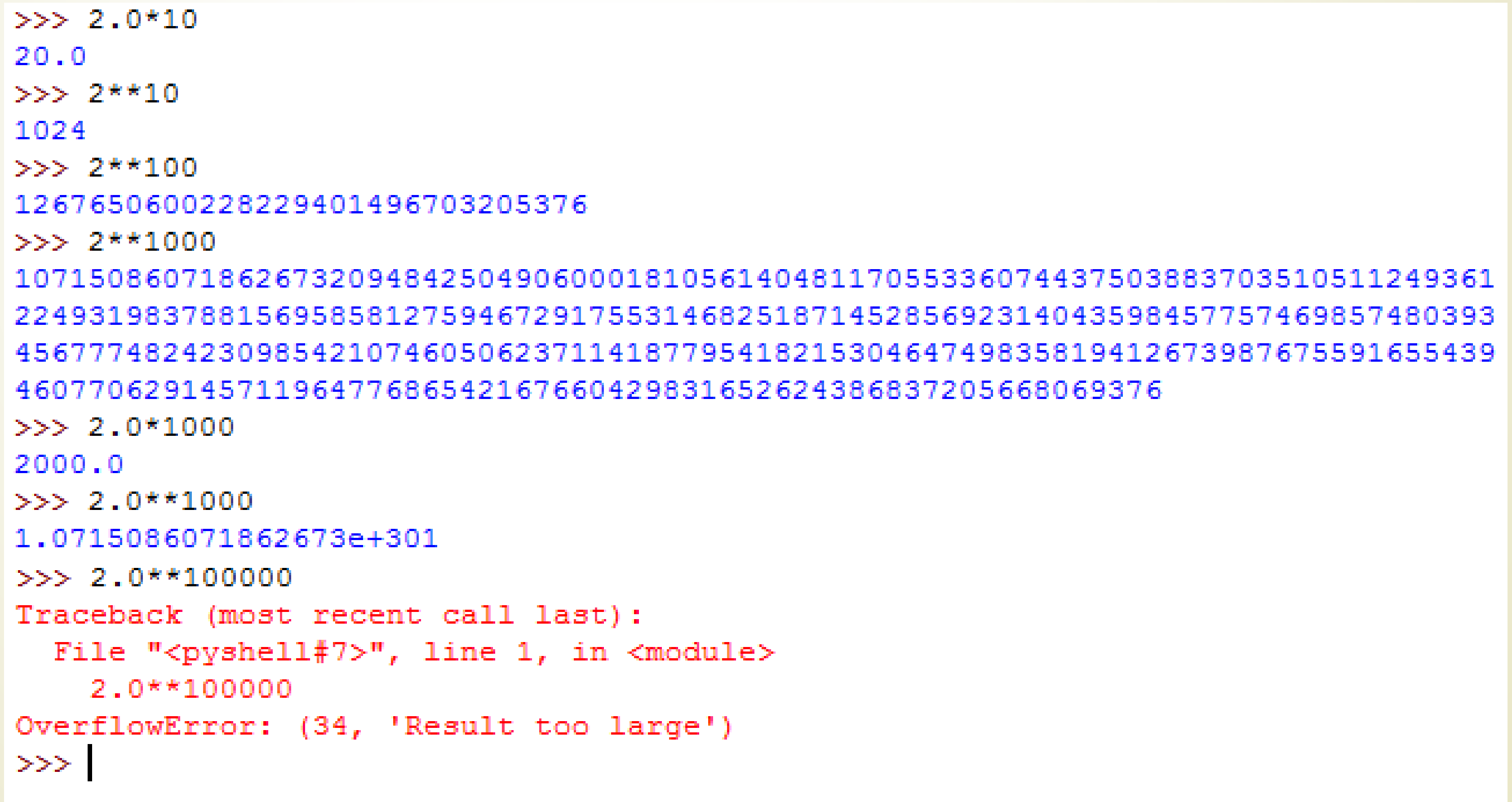
Concatenate x and y

Duplicate x three times

Duplicate x three times

Expressions

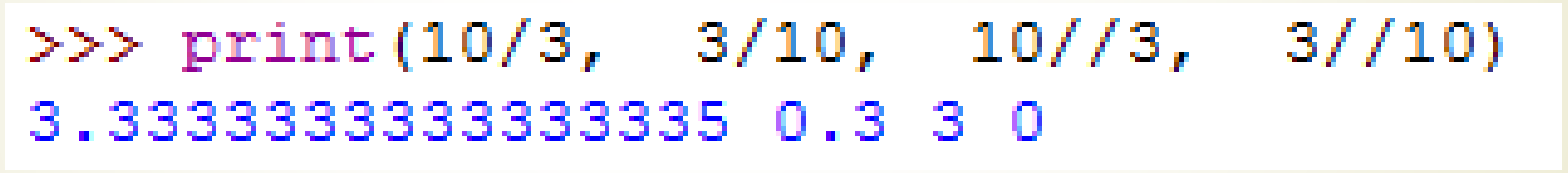
- All the arithmetic operators are subject to the limitations of the data types on which they operate.



```
>>> 2.0*10
20.0
>>> 2**10
1024
>>> 2**100
1267650600228229401496703205376
>>> 2**1000
10715086071862673209484250490600018105614048117055336074437503883703510511249361
22493198378815695858127594672917553146825187145285692314043598457757469857480393
45677748242309854210746050623711418779541821530464749835819412673987675591655439
46077062914571196477686542167660429831652624386837205668069376
>>> 2.0*1000
2000.0
>>> 2.0**1000
1.0715086071862673e+301
>>> 2.0**100000
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    2.0**100000
OverflowError: (34, 'Result too large')
>>> 
```

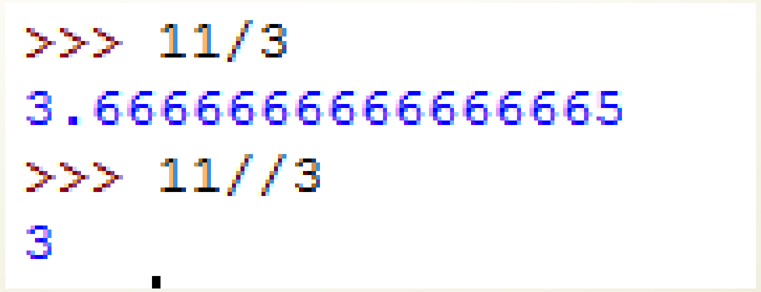
Expressions

- When we apply the +, -, *, //, %, or ** operators to two integers, the result is an integer.
- The / operator applied to two integers produces a floating-point result.



```
>>> print(10/3,  3/10,  10//3,  3//10)
3.3333333333333335 0.3 3 0
```

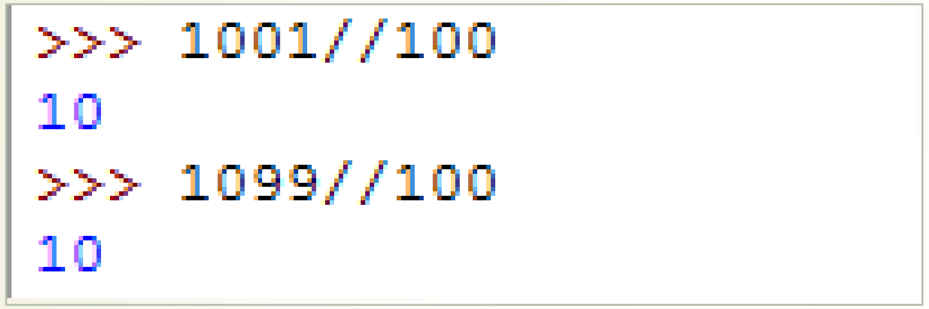
- The process of discarding the fractional part leaving only the whole number part is called *truncation*.



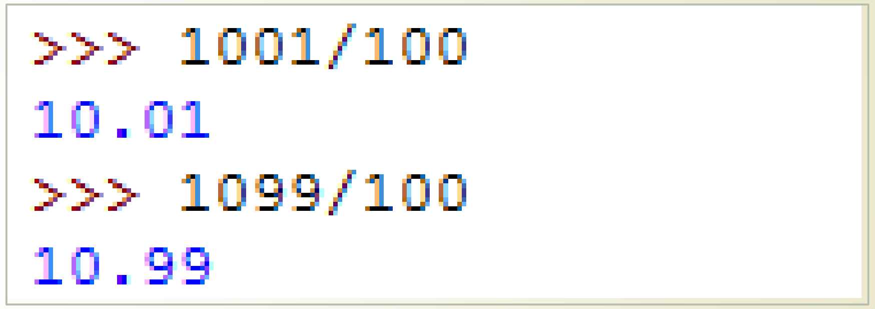
```
>>> 11/3
3.6666666666666665
>>> 11//3
3
```

Expressions

- Truncation is not rounding.

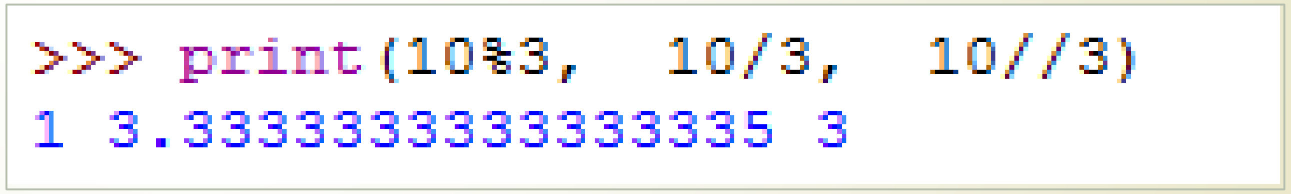


```
>>> 1001//100
10
>>> 1099//100
10
```

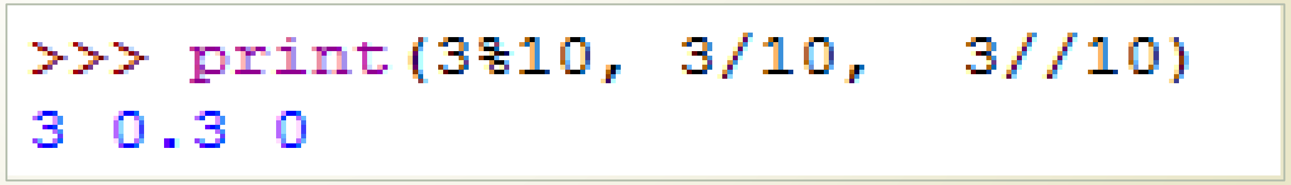


```
>>> 1001/100
10.01
>>> 1099/100
10.99
```

- The modulus operator (%) computes the remainder of integer division; thus,



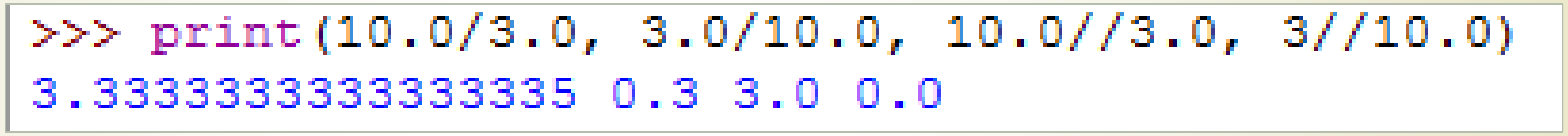
```
>>> print(10%3,  10/3,  10//3)
1 3.3333333333333335 3
```



```
>>> print(3%10, 3/10,  3//10)
3 0.3 0
```

Expressions

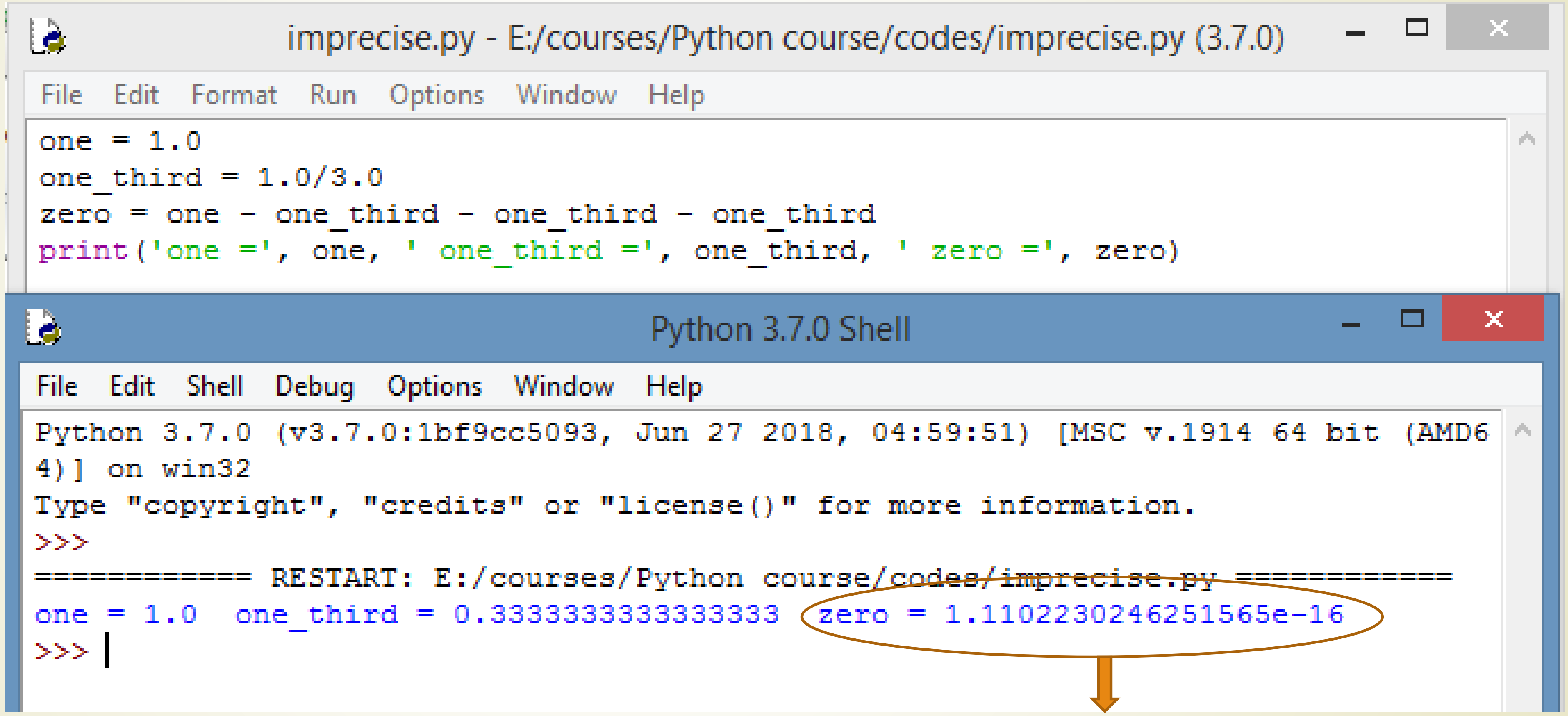
- Floating-point arithmetic always produces a floating-point result.



```
>>> print(10.0/3.0, 3.0/10.0, 10.0//3.0, 3//10.0)
3.3333333333333335 0.3 3.0 0.0
```

- Integers can be represented exactly, but floating-point numbers are imprecise approximations of real numbers.
- Floating-point numbers are not real numbers, so the result cannot be represented exactly without infinite precision.

Expressions


imprecise.py - E:/courses/Python course/codes/imprecise.py (3.7.0)
File Edit Format Run Options Window Help
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help

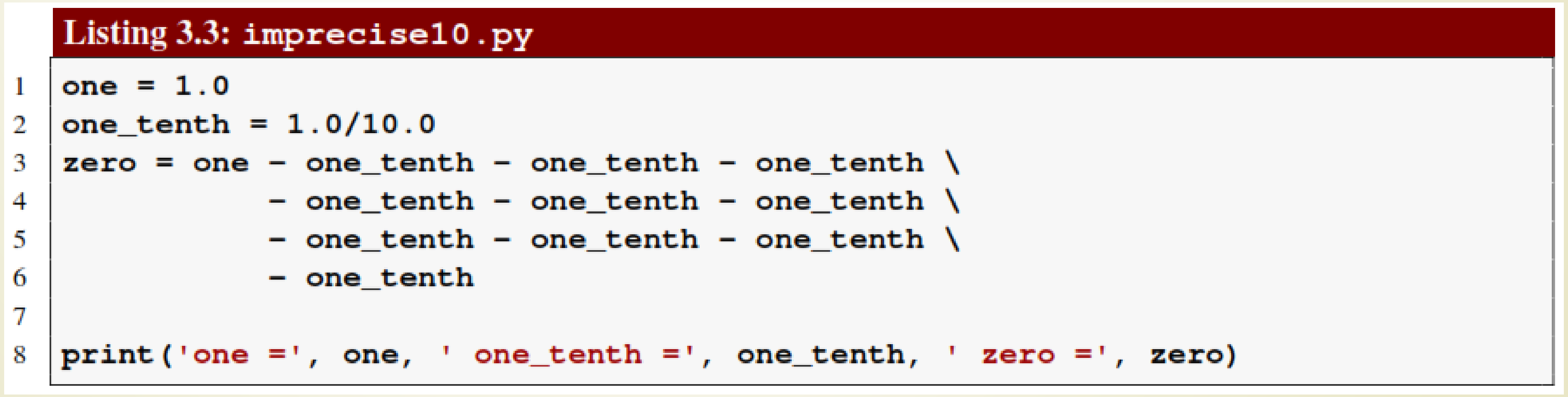
```
one = 1.0
one_third = 1.0/3.0
zero = one - one_third - one_third - one_third
print('one =', one, ' one_third =', one_third, ' zero =', zero)
```

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
============ RESTART: E:/courses/Python course/codes/imprecise.py ============
one = 1.0  one_third = 0.3333333333333333  zero = 1.1102230246251565e-16
>>> 
```

Numerical error

Expressions

- In Listing 3.3 (imprecise10.py) lines 3–6 make up a single Python statement.
- Ordinarily a Python statement ends at the end of the source code line.
- When the interpreter is processing a line that ends with a \, it automatically joins the line that follows.

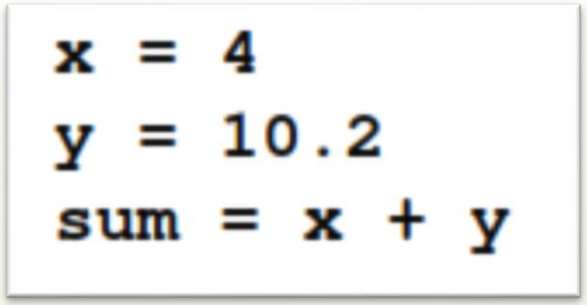


Listing 3.3: imprecise10.py

```
1  one = 1.0
2  one_tenth = 1.0/10.0
3  zero = one - one_tenth - one_tenth - one_tenth \
4             - one_tenth - one_tenth - one_tenth \
5             - one_tenth - one_tenth - one_tenth \
6             - one_tenth
7
8  print('one =', one, ' one_tenth =', one_tenth, ' zero =', zero)
```

Expressions

- Expressions may contain mixed elements; integer and floating-point.
- When an operator has mixed operands—one operand an integer and the other a floating-point number—the interpreter treats the integer operand as floating-point number and performs floating-point arithmetic.



```
x = 4
y = 10.2
sum = x + y
```

Operator Precedence and Associativity

- When different operators appear in the same expression, the normal rules of arithmetic apply. All Python operators have a precedence and associativity:
  - *Precedence*—when an expression contains two different kinds of operators, which should be applied first?
  - *Associativity*—when an expression contains two operators with the same precedence, which should be applied first?
- As in normal arithmetic, multiplication and division in Python have equal importance and are performed before addition and subtraction. We say multiplication and division have precedence over addition and subtraction.

Operator Precedence and Associativity

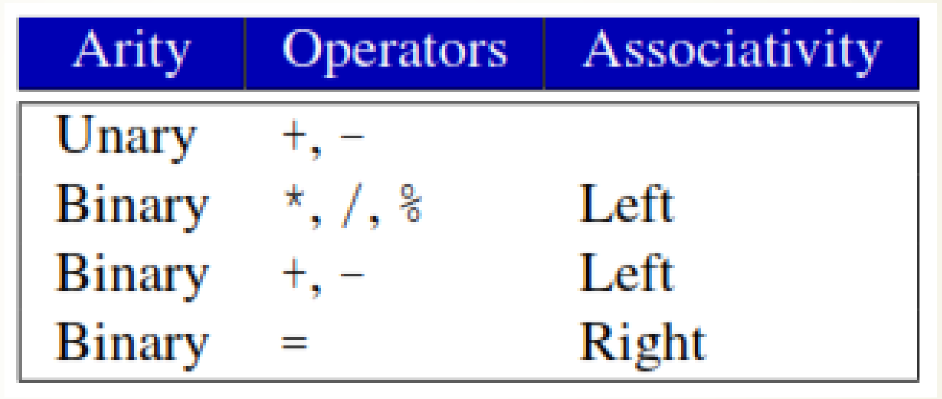
- The multiplicative operators (*, /, //, and %) have equal precedence with each other, and the additive operators (binary + and -) have equal precedence with each other.

The multiplicative operators have precedence over the additive operators.

- As in standard arithmetic, in Python if the addition is to be performed first, parentheses can override the precedence rules.
- Multiple sets of parentheses can be arranged and nested in any ways that are acceptable in standard arithmetic.

Operator Precedence and Associativity

Table 3.2: Operator precedence and associativity.

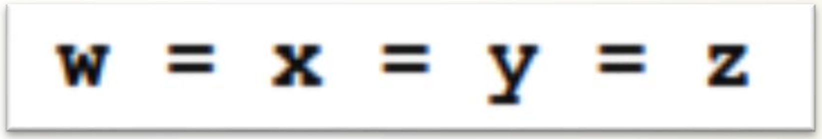


Arity	Operators	Associativity
Unary	+, −	
Binary	*, /, %	Left
Binary	+, −	Left
Binary	=	Right

- The operators in each row have a higher precedence than the operators below it.
- Operators within a row have the same precedence.

Operator Precedence and Associativity

- The assignment operator supports a technique known as chained assignment.
- The code

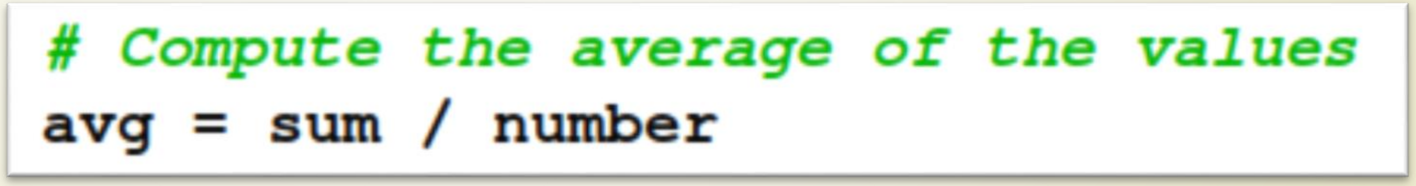


```
w = x = y = z
```

  should be read right to left.
- As in the case of precedence, parentheses can be used to override the natural associativity within an expression.
- The unary operators have a higher precedence than the binary operators, and the unary operators are right associative.

Comments

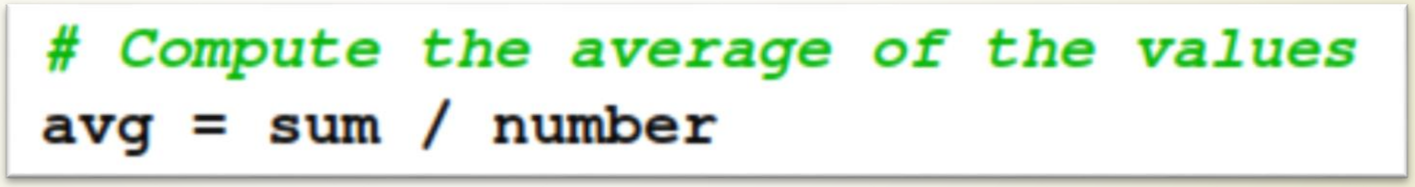
- Good programmers annotate their code by inserting remarks that explain the purpose of a section of code or why they chose to write a section of code the way they did. These notes are meant for human readers, not the interpreter. It is common in industry for programs to be reviewed for correctness by other programmers or technical managers.
- Any text contained within comments is ignored by the Python interpreter. The # symbol begins a comment in the source code.
- The comment is in effect until the end of the line of code:



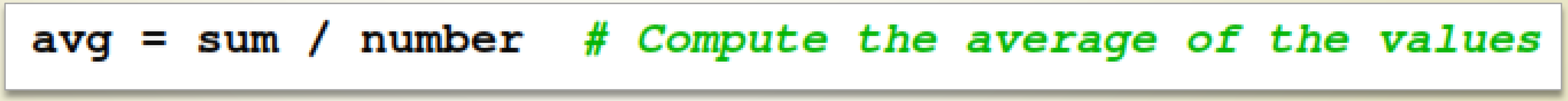
```
# Compute the average of the values
avg = sum / number
```

Comments

- Good programmers annotate their code by inserting remarks that explain the purpose of a section of code or why they chose to write a section of code the way they did.
- Any text contained within comments is ignored by the Python interpreter. The # symbol begins a comment in the source code.
- The comment is in effect until the end of the line of code:



```
# Compute the average of the values
avg = sum / number
```



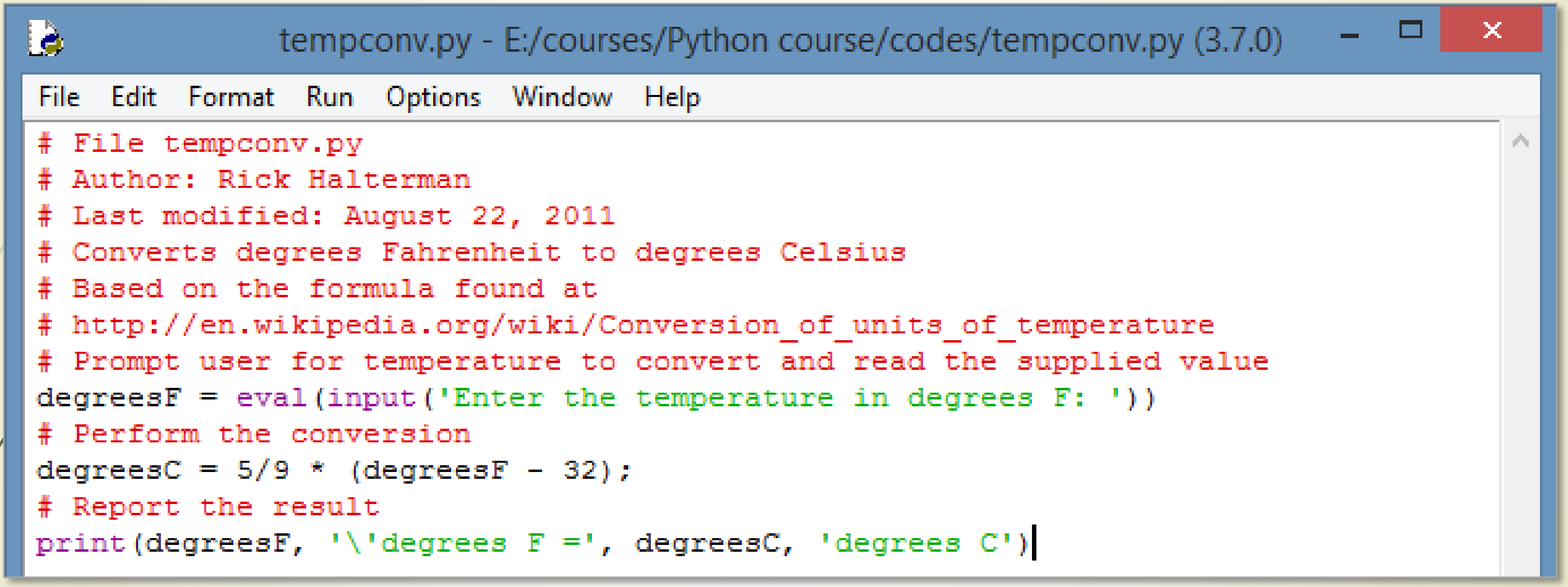
```
avg = sum / number  # Compute the average of the values
```

Errors

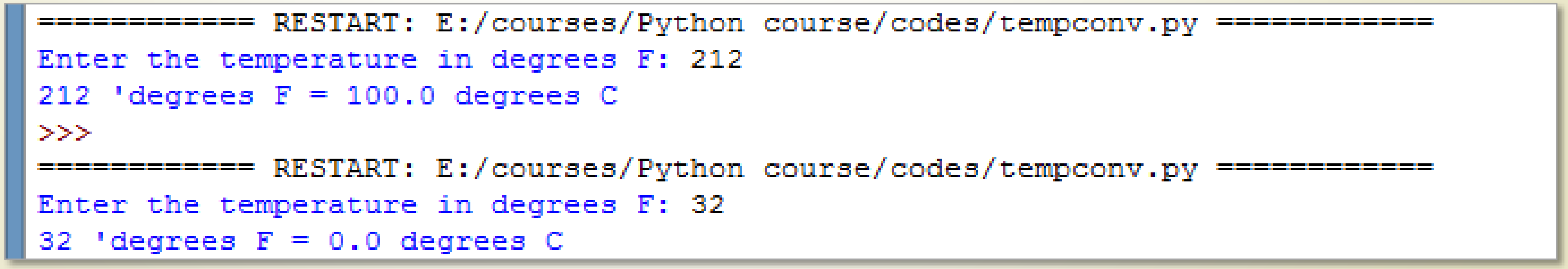
In Python, there are three general kinds of errors:

- syntax errors, (*translation phase*)
- run-time errors, and
- logic errors.

Arithmetic Examples

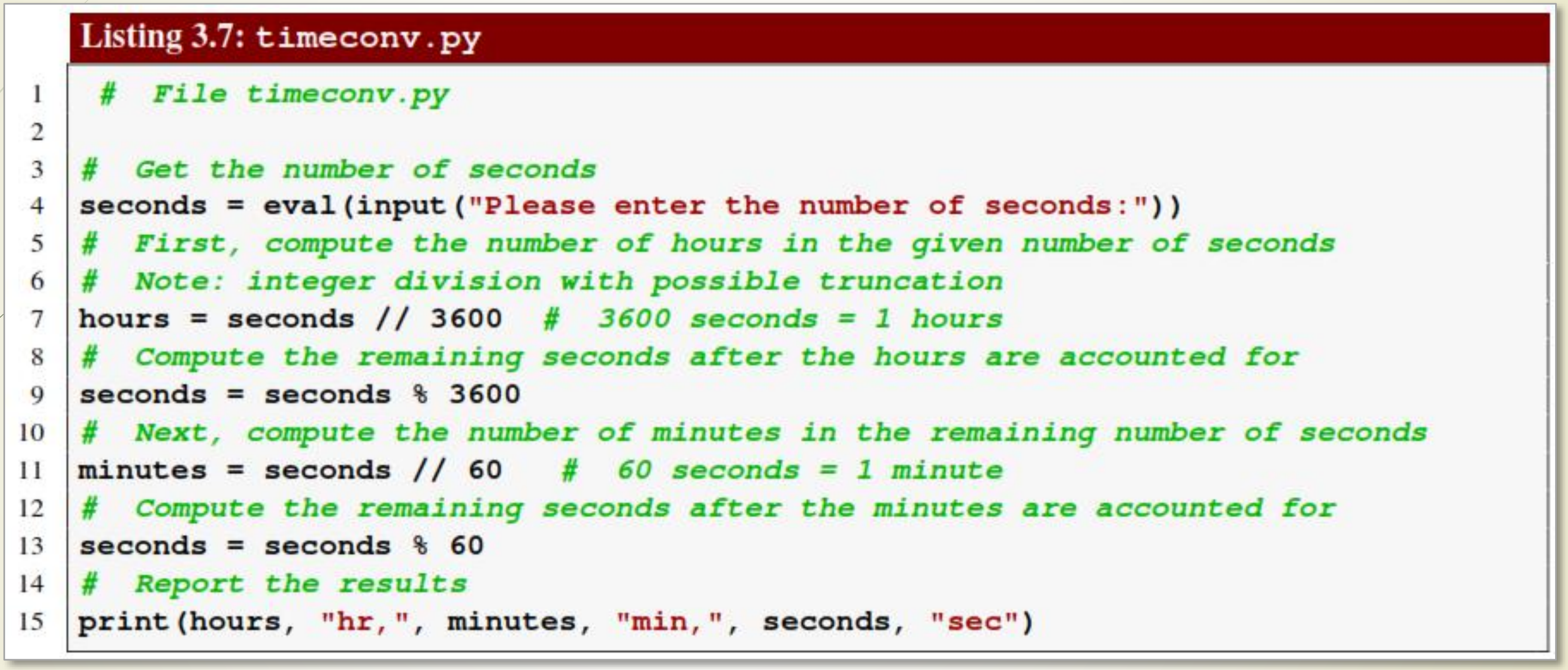

tempconv.py - E:/courses/Python course/codes/tempconv.py (3.7.0)
File Edit Format Run Options Window Help

```
# File tempconv.py
# Author: Rick Halterman
# Last modified: August 22, 2011
# Converts degrees Fahrenheit to degrees Celsius
# Based on the formula found at
# http://en.wikipedia.org/wiki/Conversion_of_units_of_temperature
# Prompt user for temperature to convert and read the supplied value
degreesF = eval(input('Enter the temperature in degrees F: '))
# Perform the conversion
degreesC = 5/9 * (degreesF - 32);
# Report the result
print(degreesF, '\'degrees F =', degreesC, 'degrees C')
```



```
============ RESTART: E:/courses/Python course/codes/tempconv.py ============
Enter the temperature in degrees F: 212
212 'degrees F = 100.0 degrees C
>>>
============ RESTART: E:/courses/Python course/codes/tempconv.py ============
Enter the temperature in degrees F: 32
32 'degrees F = 0.0 degrees C
```

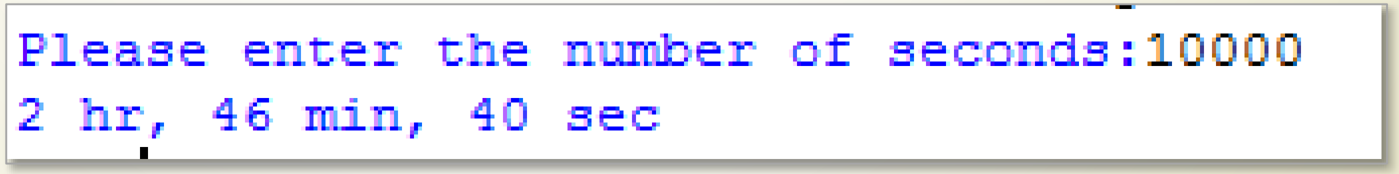
Arithmetic Examples



Listing 3.7: timeconv.py

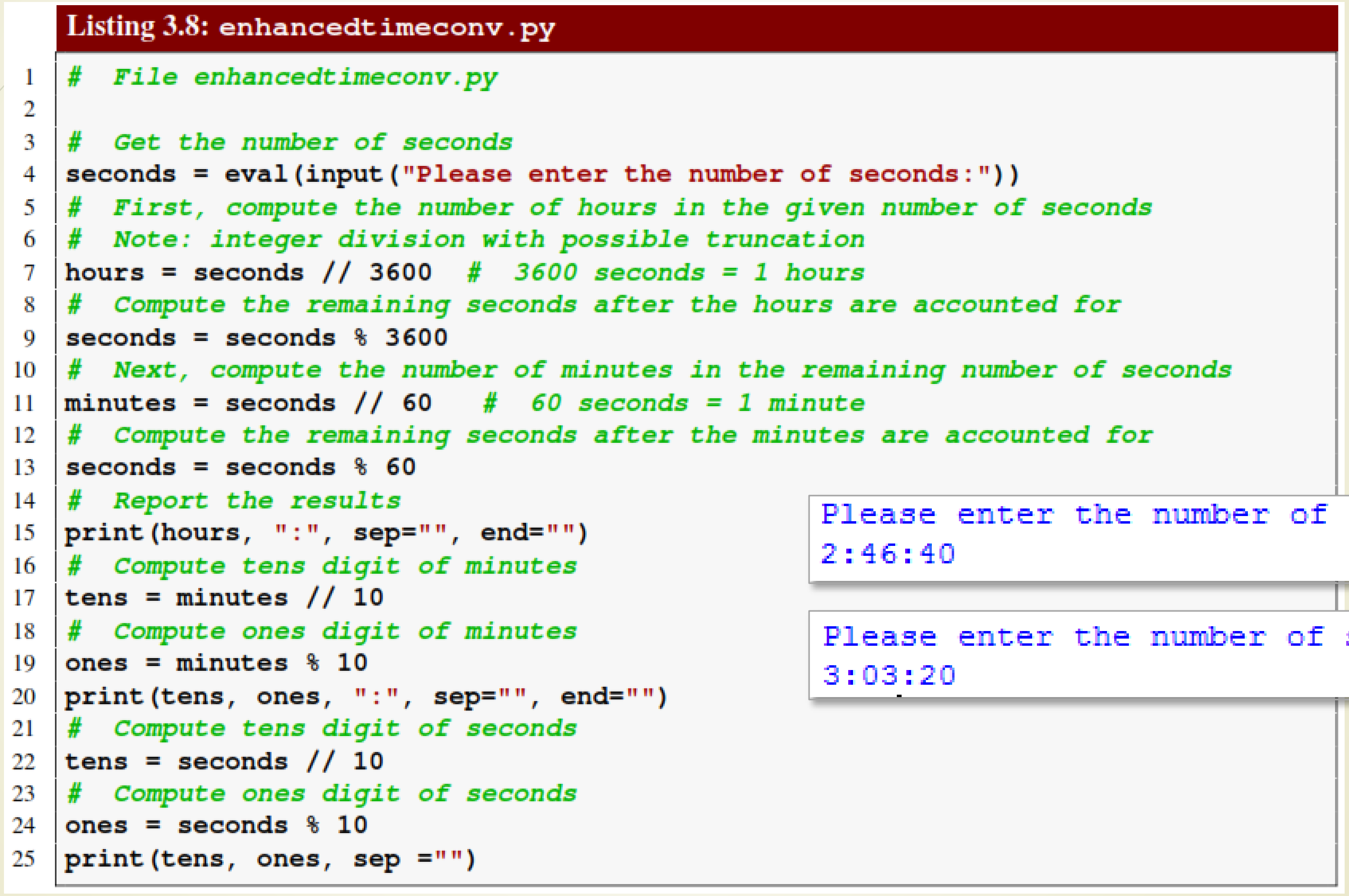
```python
 #  File timeconv.py

#  Get the number of seconds
seconds = eval(input("Please enter the number of seconds:"))
#  First, compute the number of hours in the given number of seconds
#  Note: integer division with possible truncation
hours = seconds // 3600   #  3600 seconds = 1 hours
#  Compute the remaining seconds after the hours are accounted for
seconds = seconds % 3600
#  Next, compute the number of minutes in the remaining number of seconds
minutes = seconds // 60    #  60 seconds = 1 minute
#  Compute the remaining seconds after the minutes are accounted for
seconds = seconds % 60
#  Report the results
print(hours, "hr,", minutes, "min,", seconds, "sec")
```



```
Please enter the number of seconds:10000
2 hr, 46 min, 40 sec
```

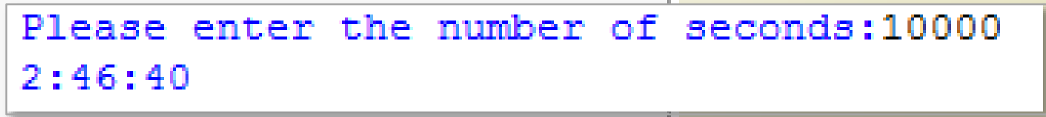
Arithmetic Examples



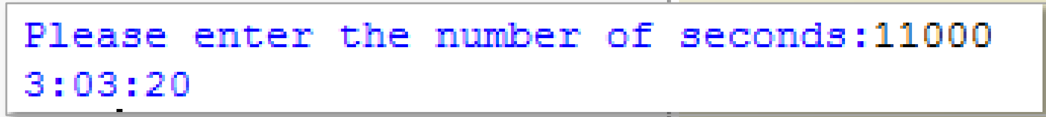
Listing 3.8: `enhancedtimeconv.py`

```python
#  File enhancedtimeconv.py

#  Get the number of seconds
seconds = eval(input("Please enter the number of seconds:"))
#  First, compute the number of hours in the given number of seconds
#  Note: integer division with possible truncation
hours = seconds // 3600   #  3600 seconds = 1 hours
#  Compute the remaining seconds after the hours are accounted for
seconds = seconds % 3600
#  Next, compute the number of minutes in the remaining number of seconds
minutes = seconds // 60   #  60 seconds = 1 minute
#  Compute the remaining seconds after the minutes are accounted for
seconds = seconds % 60
#  Report the results
print(hours, ":", sep="", end="")
#  Compute tens digit of minutes
tens = minutes // 10
#  Compute ones digit of minutes
ones = minutes % 10
print(tens, ones, ":", sep="", end="")
#  Compute tens digit of seconds
tens = seconds // 10
#  Compute ones digit of seconds
ones = seconds % 10
print(tens, ones, sep ="")
```



```
Please enter the number of seconds:10000
2:46:40
```



```
Please enter the number of seconds:11000
3:03:20
```

More Arithmetic Operators

- Any statement of the form *x op= exp*

 is equivalent to *x = x op exp;*

where

- x is a variable.
- *op*= is an arithmetic operator combined with the assignment operator; for our purposes, the ones most useful to us are +=, -=, *=, /=, //=, and %=.
- *exp* is an expression compatible with the variable

```
x = x + 1
```

$\longrightarrow$

```
x += 1
```

```
x -= 1      #  Same as x = x - 1;
```

More Arithmetic Operators

Similarly;

```
x *= y + z;
```

```
x = x * (y + z);
```

The version using the arithmetic assignment does not require parentheses. The arithmetic assignment is especially handy if a variable with a long name must be modified; consider

```
temporary_filename_length = temporary_filename_length / (y + z);
```

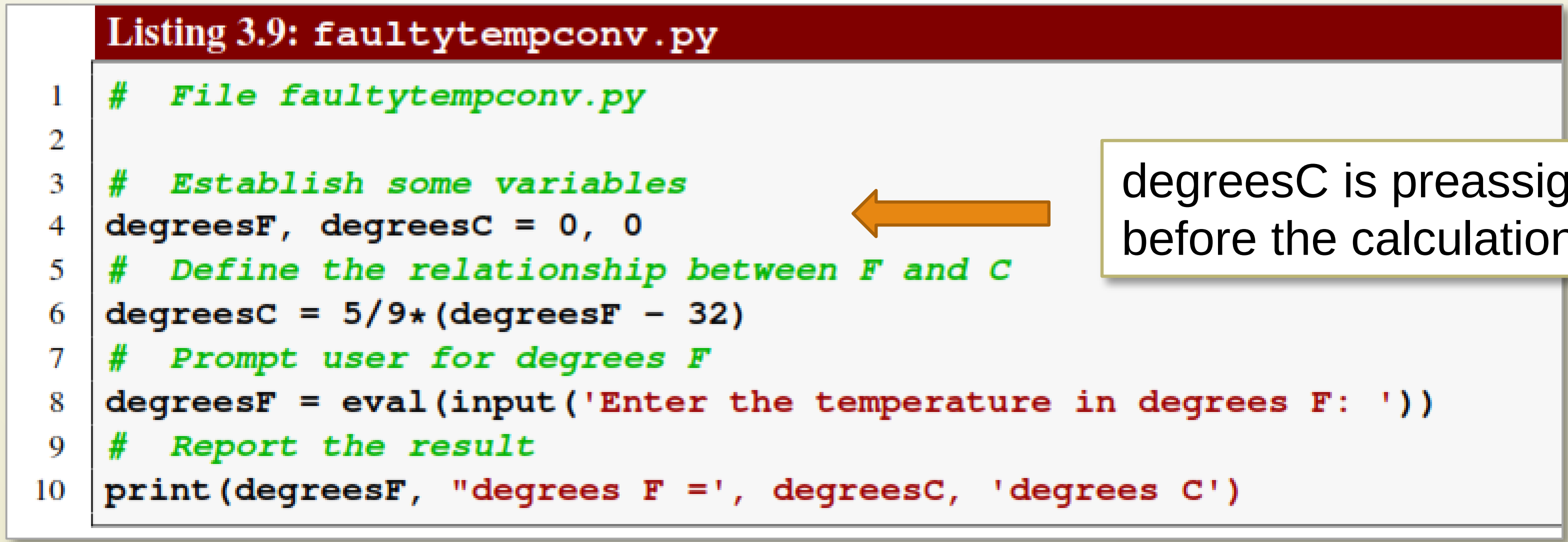
Versus

```
temporary_filename_length /= y + z;
```

Do not accidentally reverse the order of the symbols for the arithmetic assignment operators

Algorithms

- An algorithm is a finite sequence of steps, each step taking a finite length of time, that solves a problem or computes a result.
- A computer program is one example of an algorithm.
- The ordering of steps is very important in a computer program.



Listing 3.9: faultytempconv.py

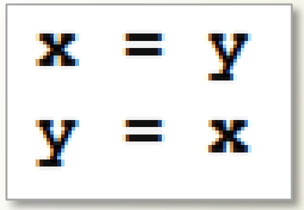
```
#  File faultytempconv.py

#  Establish some variables
degreesF, degreesC = 0, 0
#  Define the relationship between F and C
degreesC = 5/9*(degreesF - 32)
#  Prompt user for degrees F
degreesF = eval(input('Enter the temperature in degrees F: '))
#  Report the result
print(degreesF, "degrees F =', degreesC, 'degrees C')
```

degreesC is preassigned before the calculation

Algorithms

- As another example, suppose x and y are two variables in some program.
- How would we interchange the values of the two variables? We want x to have y's original value and y to have x's original value.
- This code may seem reasonable:

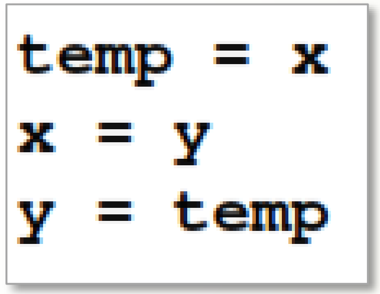


```
x = y
y = x
```

- The problem with this section of code is that after the first statement is executed, x and y both have the same value (y's original value). The second assignment is superfluous and does nothing to change the values of x or y.

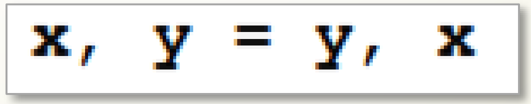
Algorithms

- The solution requires a third variable to remember the original value of one the variables before it is reassigned. The correct code to swap the values is



```
temp = x
x = y
y = temp
```

- We can use *tuple* assignment to make the swap even simpler:



```
x, y = y, x
```

Thank you