Exposure CS 2021 for CS1

Chapter 4 Section 2-7 Slides

Operators, Simple Data Types, Shortcuts and Documentation

PowerPoint Presentation created by:
Mr. John L. M. Schram
and Mr. Leon Schram
Authors of Exposure
Computer Science

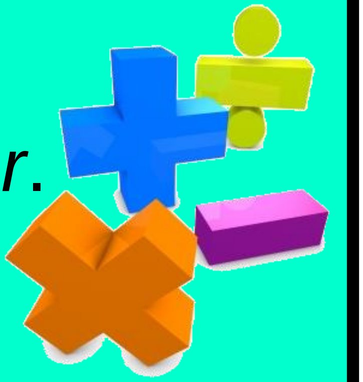
Section 4.2

Arithmetic Operations

Operations, Operators and Operands

2 + 3 is an addition *operation*.

The plus sign (**+**) is the addition *operator*.

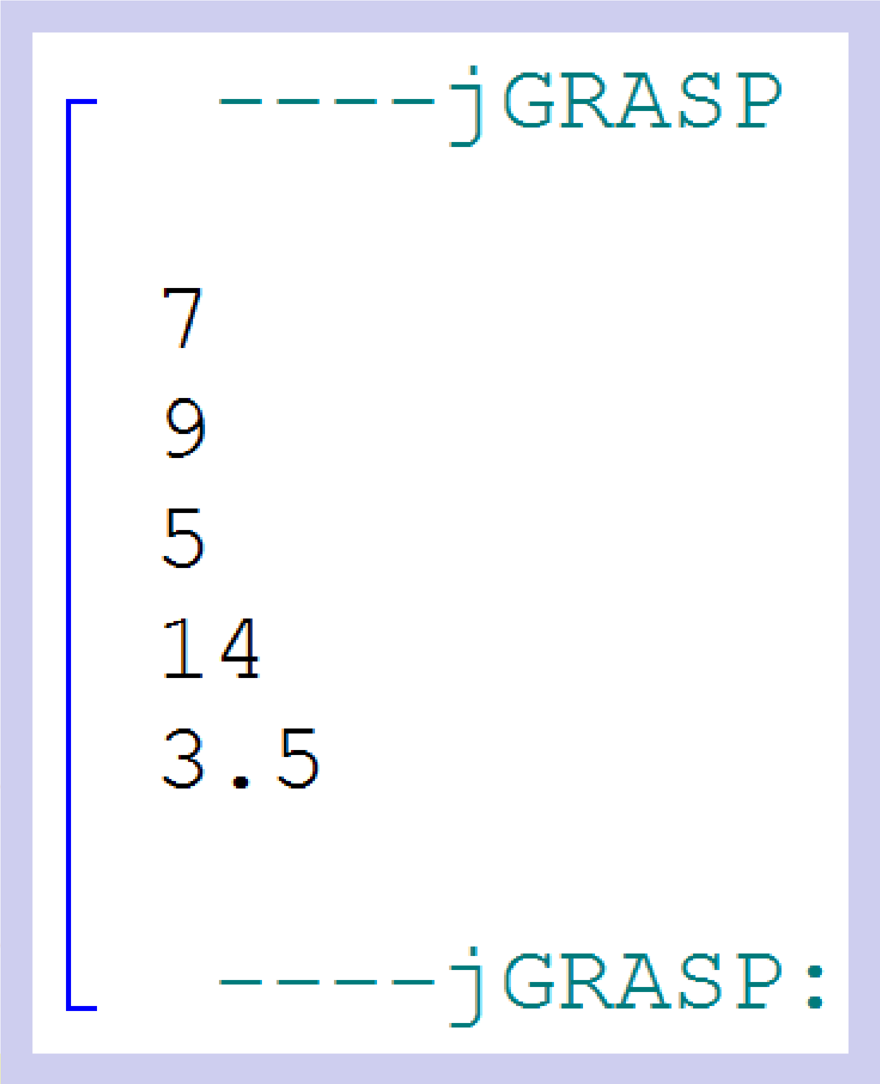


The **2** and the **3** are the *operands*.

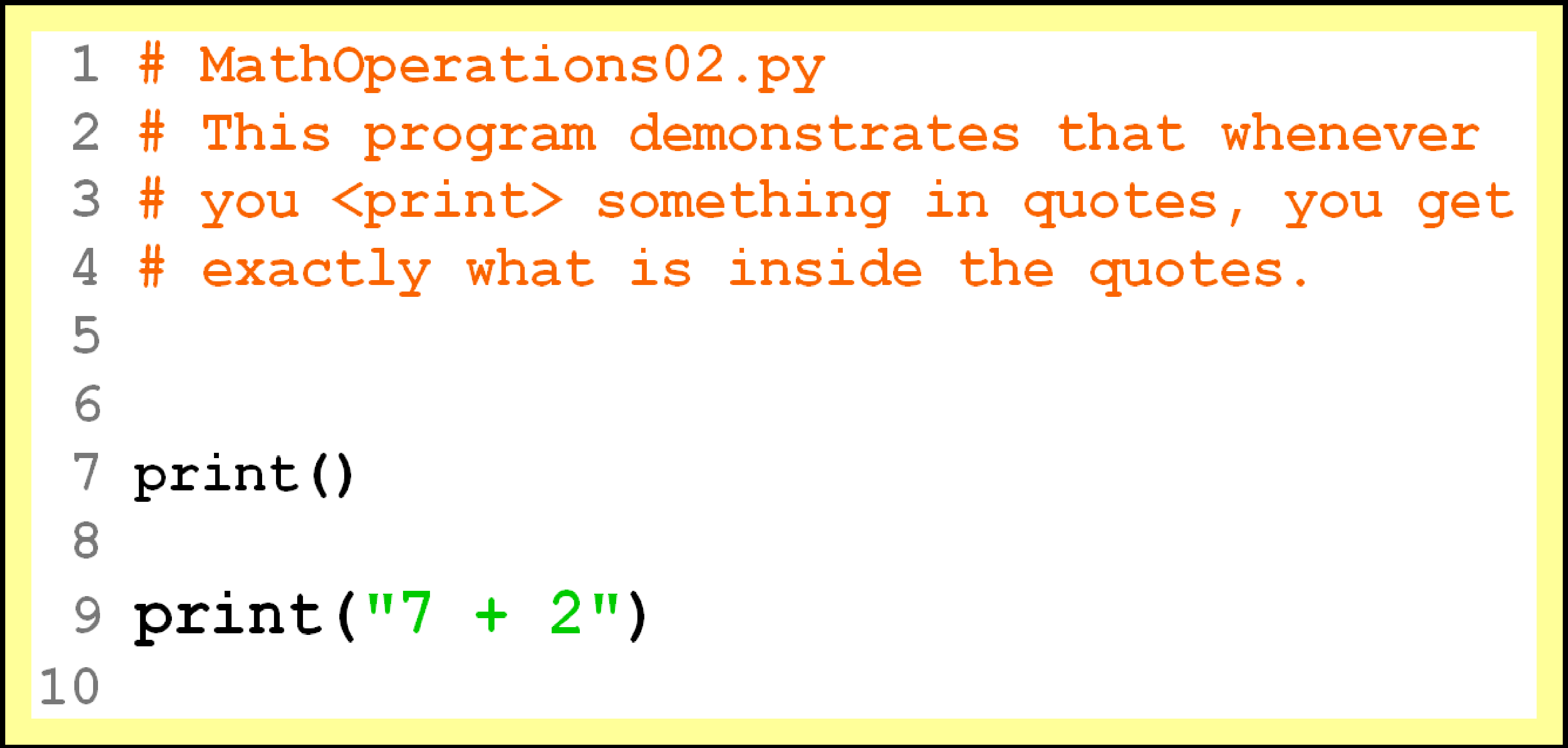
When you have 2 operands and 1 arithmetic operator, you have a *Binary Arithmetic Operation*.

```
# MathOperations01.py
# This program demonstrates that the <print>
# command can evaluate and display the result
# of mathematical expressions.


print()

print(7)

print(7 + 2)

print(7 - 2)

print(7 * 2)     # Multiplication

print(7 / 2)     # Division

```

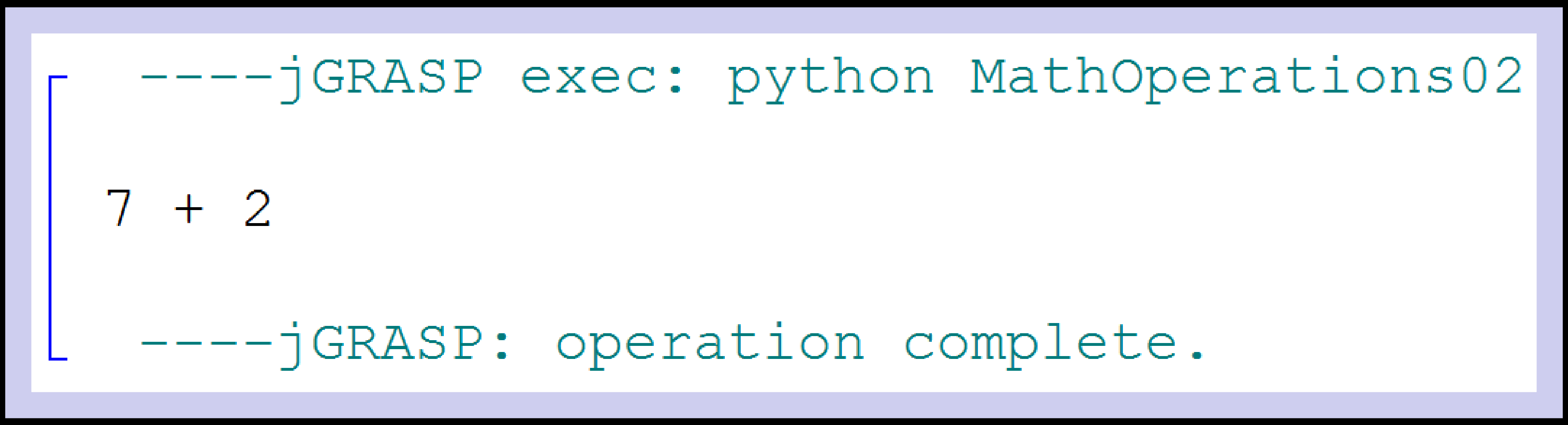
```python
# MathOperations01.py
# This program demonstrates that the <print>
# command can evaluate and display the result
# of mathematical expressions.


print()

print(7)

print(7 + 2)

print(7 - 2)

print(7 * 2)

print(7 / 2)

```



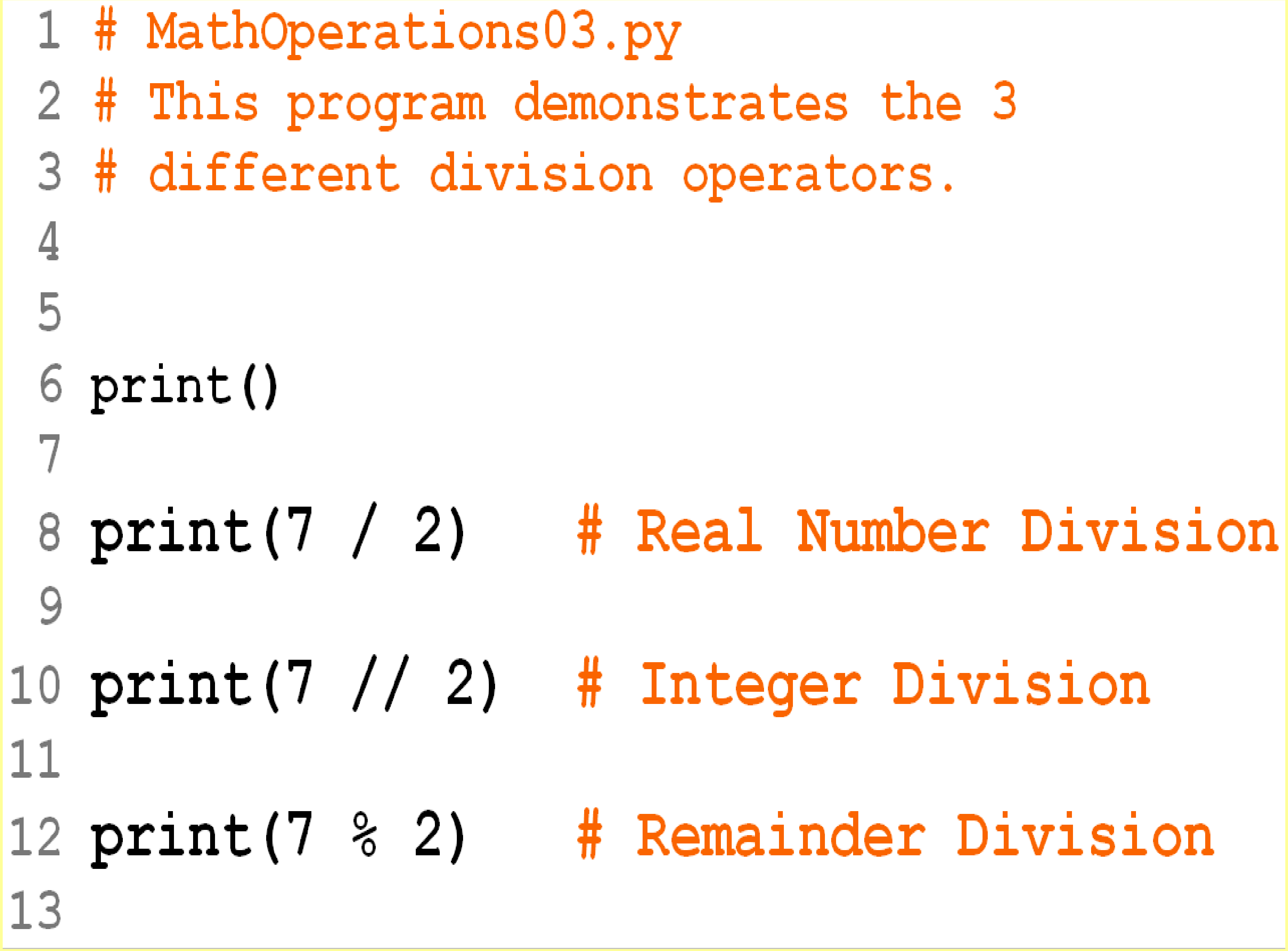
```
----jGRASP


7
9
5
14
3.5

----jGRASP:
```



```python
# MathOperations02.py
# This program demonstrates that whenever
# you <print> something in quotes, you get
# exactly what is inside the quotes.


print()

print("7 + 2")

```

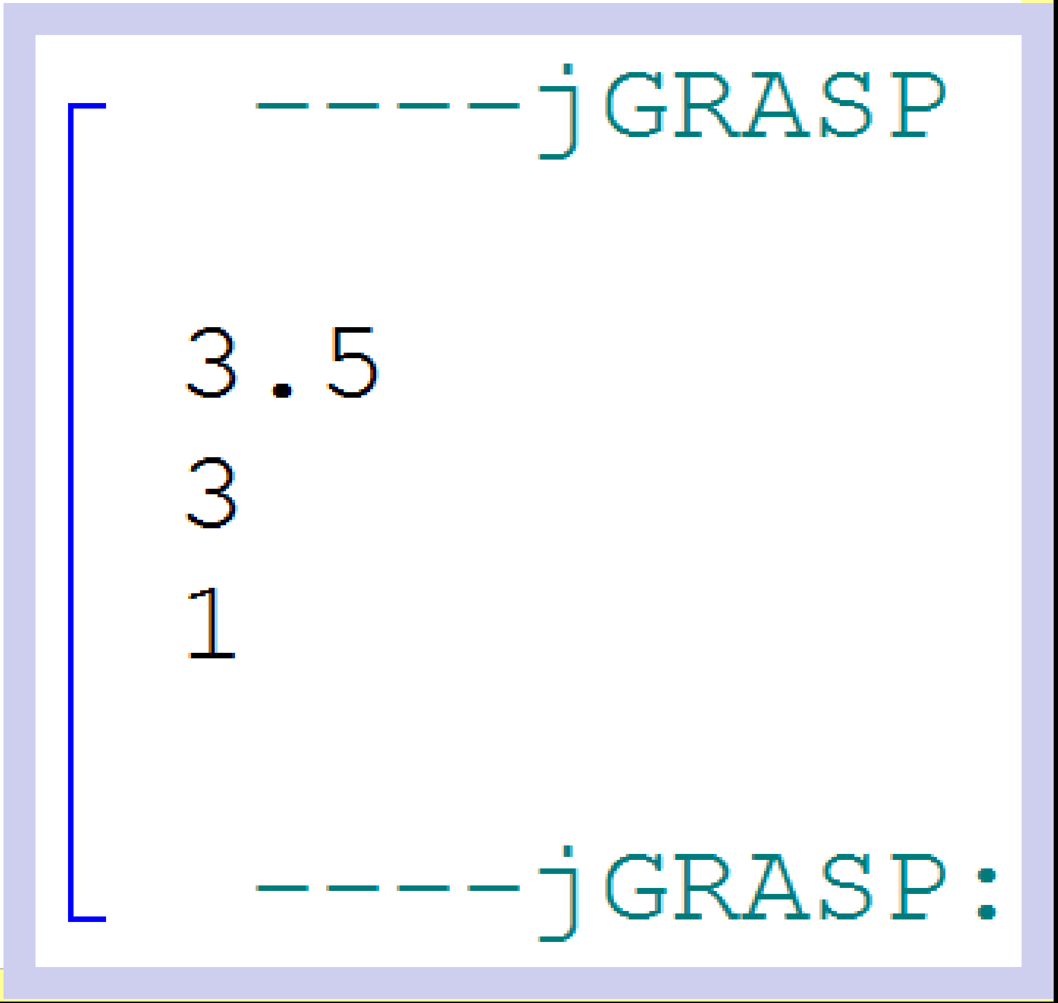


```
 ----jGRASP exec: python MathOperations02

7 + 2

 ----jGRASP: operation complete.
```



```python
# MathOperations03.py
# This program demonstrates the 3
# different division operators.


print()

print(7 / 2)    # Real Number Division

print(7 // 2)   # Integer Division

print(7 % 2)    # Remainder Division

```

```python
# MathOperations03.py
# This program demonstrates the 3
# different division operators.


print()

print(7 / 2)

print(7 // 2)

print(7 % 2)

```



```
 ----jGRASP


3.5
3
1


 ----jGRASP:
```

Integer Division Examples

```
12 // 1  = 12
12 // 2  = 6
12 // 3  = 4
12 // 4  = 3
12 // 5  = 2
12 // 6  = 2
12 // 7  = 1
```

```
12 // 8  = 1
12 // 9  = 1
12 // 10 = 1
12 // 11 = 1
12 // 12 = 1
12 // 13 = 0
12 // 0  = undefined
```

Remainder Division Examples

`12 % 1  = 0`	`12 % 8  = 4`
`12 % 2  = 0`	`12 % 9  = 3`
`12 % 3  = 0`	`12 % 10 = 2`
`12 % 4  = 0`	`12 % 11 = 1`
`12 % 5  = 2`	`12 % 12 = 0`
`12 % 6  = 0`	`12 % 13 = 12`
`12 % 7  = 5`	`12 % 0  =` *undefined*

What do the **blue** numbers have in common?

12 % **1** = 0

12 % **2** = 0

12 % **3** = 0

12 % **4** = 0

12 % 5 = 2

12 % **6** = 0

12 % 7 = 5

12 % 8 = 4

12 % 9 = 3

12 % 10 = 2

12 % 11 = 1

12 % **12** = 0

12 % 13 = 12

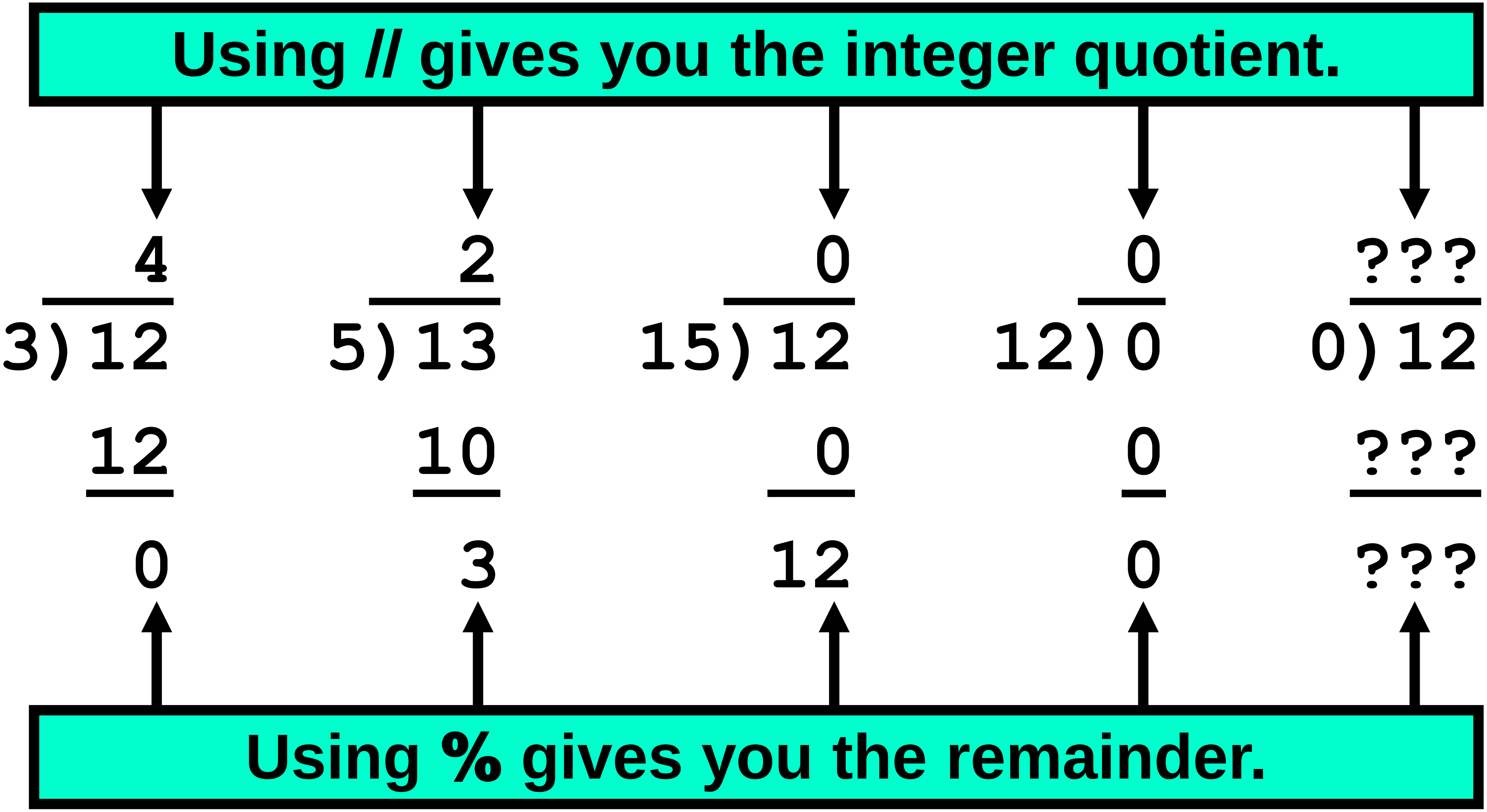
12 % 0 = *undefined*

What do the **blue** numbers have in common?

12 % **1** = 0	12 % 8 = 4
12 % **2** = 0	12 % 9 = 3
12 % **3** = 0	12 % 10 = 2
12 % **4** = 0	12 % 11 = 1
12 % 5 = 2	12 % **12** = 0
12 % **6** = 0	12 % 13 = 12
12 % 7 = 5	12 % 0 = *undefined*

They are all factors of 12.

Flashback To
Elementary School Long Division



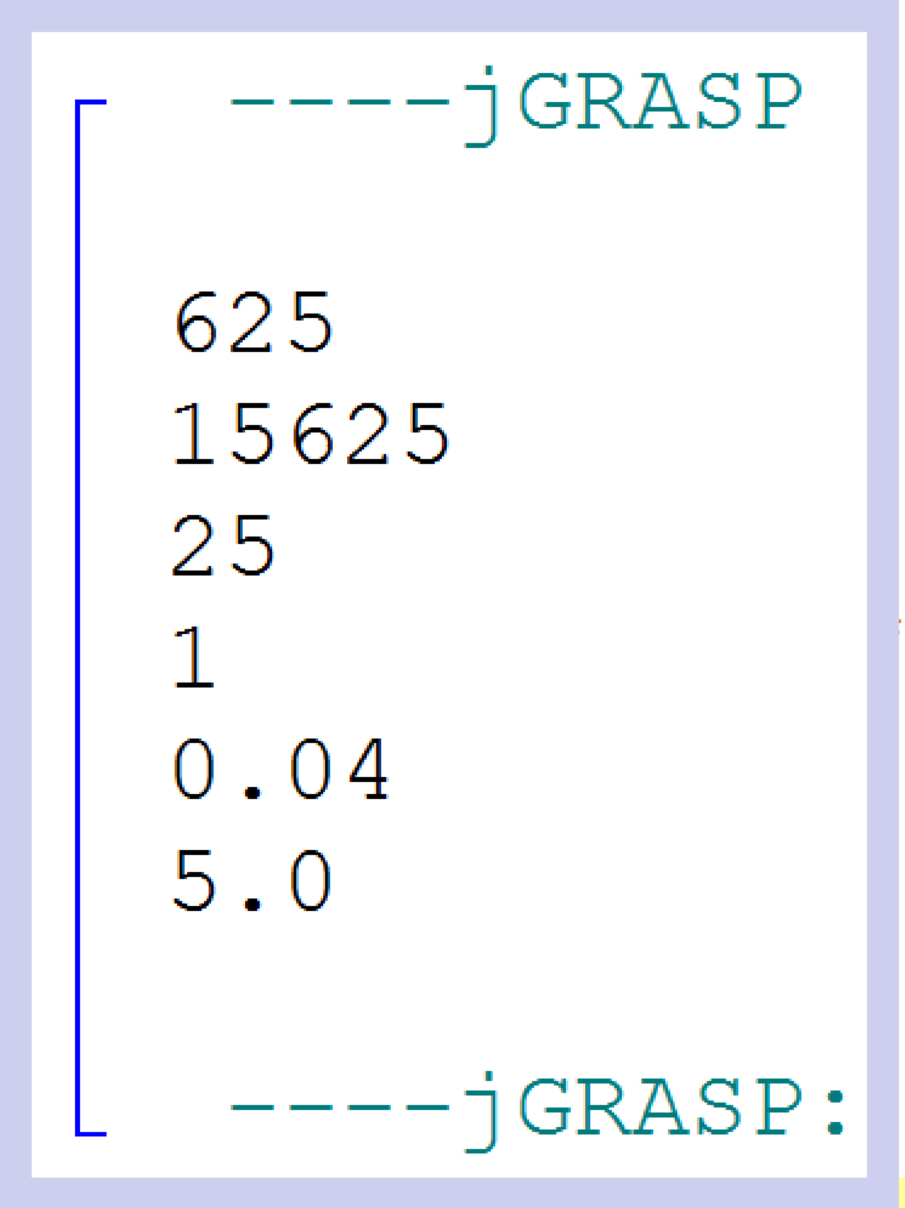
Using // gives you the integer quotient.

```
   4        2          0        0      ???
 ____     ____       ____     ____     ____
3)12     5)13      15)12     12)0     0)12
  12       10          0        0      ???
  __       __         __        _      ___
   0        3         12        0      ???
```

Using % gives you the remainder.

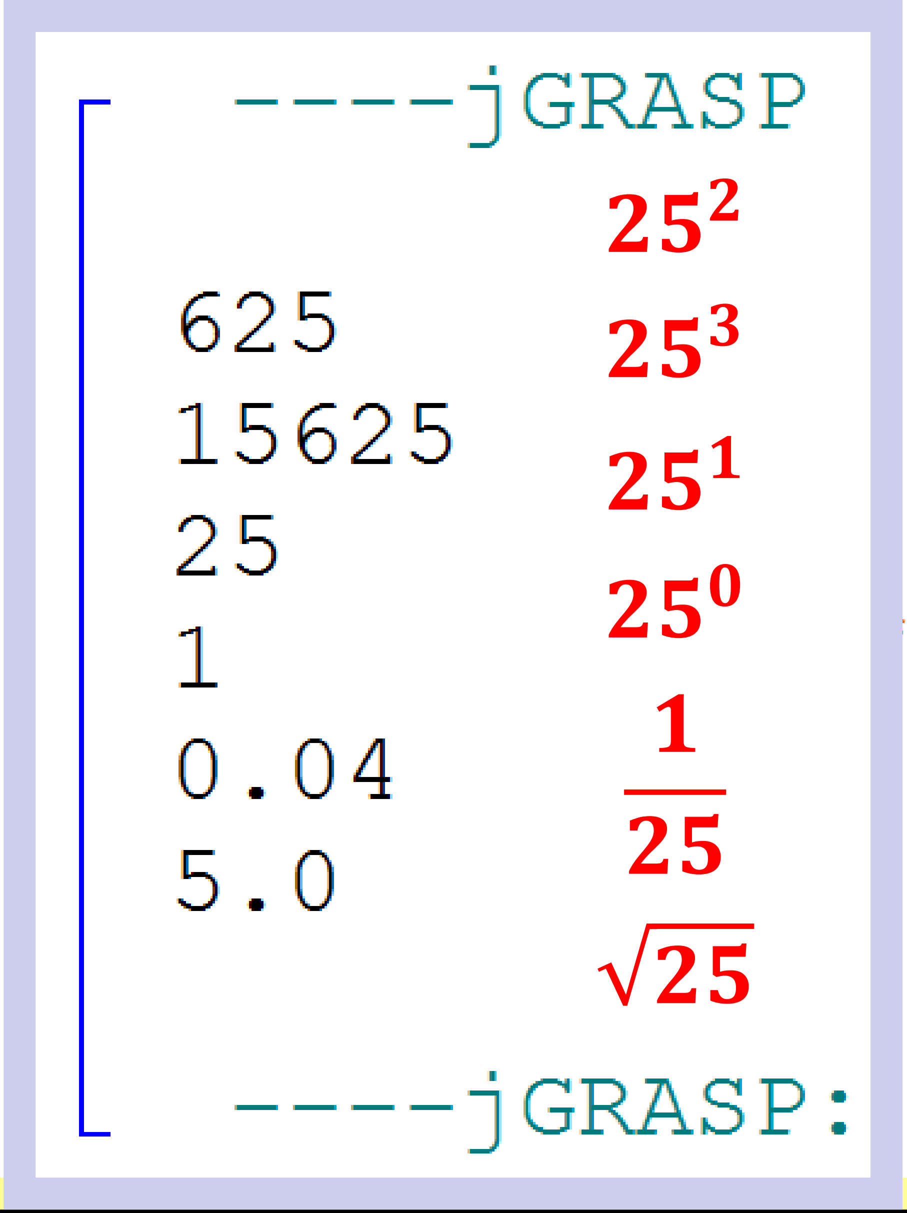
```
# MathOperations04.py
#  This program demonstrates the Exponent Operator **


print()

print(25 ** 2)      # 25 * 25

print(25 ** 3)      # 25 * 25 * 25

print(25 ** 1)      #  Anything to the power of 1 is itself

print(25 ** 0)      #  Anything to the power of 0 is 1

print(25 ** -1)     #  The reciprocal of 25 or 1/25

print(25 ** 0.5)    #  The square root of 25
```

```python
# MathOperations04.py
# This program demonstrates the Exponent Operator **


print()

print(25 ** 2)

print(25 ** 3)

print(25 ** 1)

print(25 ** 0)

print(25 ** -1)

print(25 ** 0.5)
```



```
 ----jGRASP

 625
 15625
 25
 1
 0.04
 5.0

 ----jGRASP:
```

```python
# MathOperations04.py
# This program demonstrates the Exponent Operator **


print()

print(25 ** 2)

print(25 ** 3)

print(25 ** 1)

print(25 ** 0)

print(25 ** -1)

print(25 ** 0.5)
```



```
 ----jGRASP

625
15625
25
1
0.04
5.0

 ----jGRASP:
```

25^2

25^3

25^1

25^0

$\frac{1}{25}$

$\sqrt{25}$

Python's 7 Arithmetic Operators

Addition	+
Subtraction	–
Multiplication	*
Real Number Division	/
Integer Division	//
Remainder Division	%
Exponents	**

Remember Order of Operations!

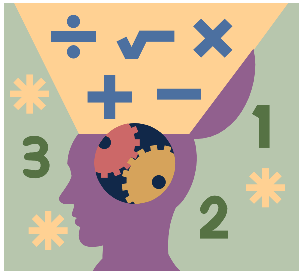
P
E
M
D
A
S

Parentheses

Exponents

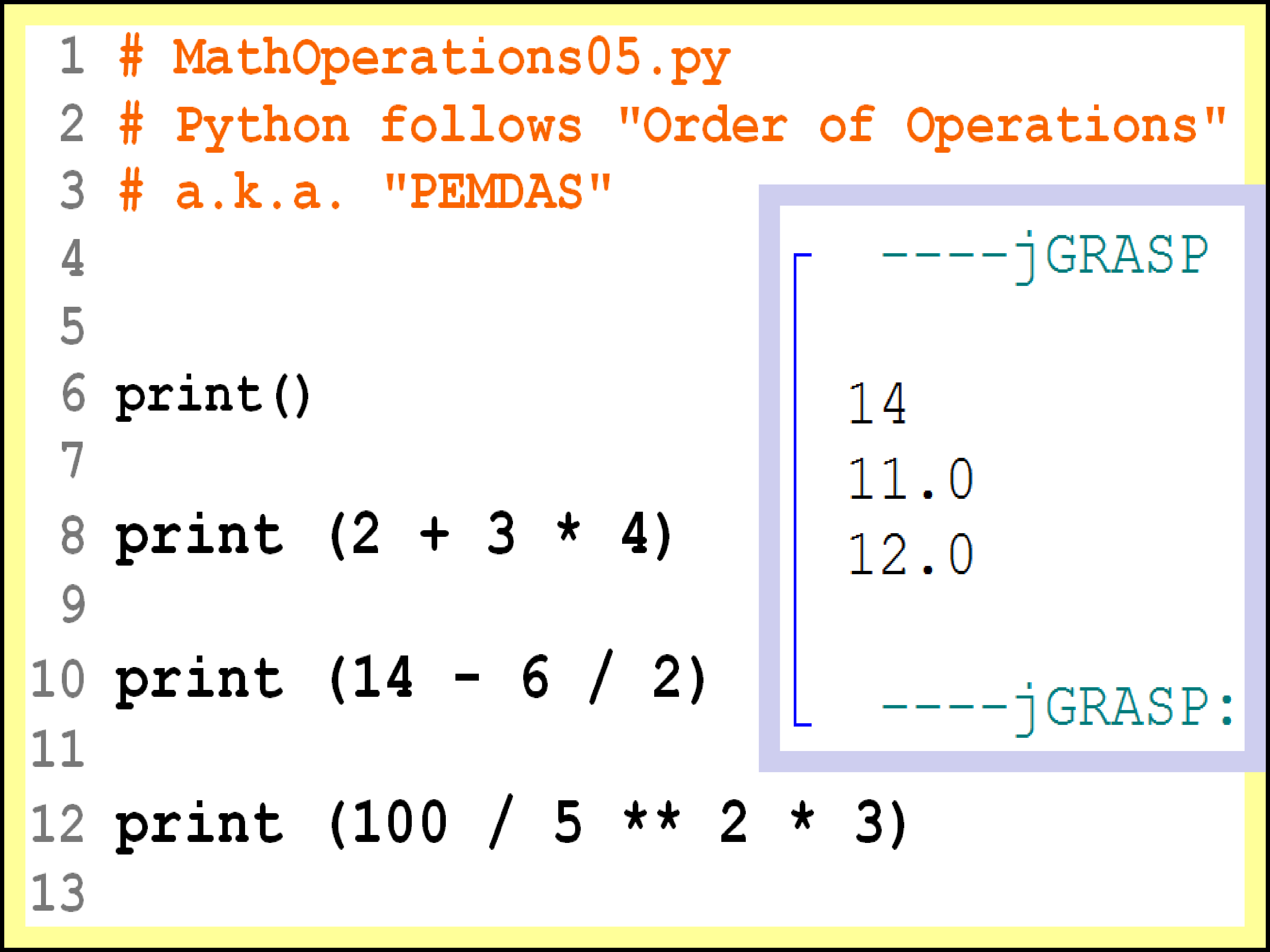
Multiplication & Division

Addition & Subtraction



"Please Excuse My Dear Aunt Sally."

```
# MathOperations05.py
# Python follows "Order of Operations"
# a.k.a. "PEMDAS"


print()

print (2 + 3 * 4)

print (14 - 6 / 2)

print (100 / 5 ** 2 * 3)

```

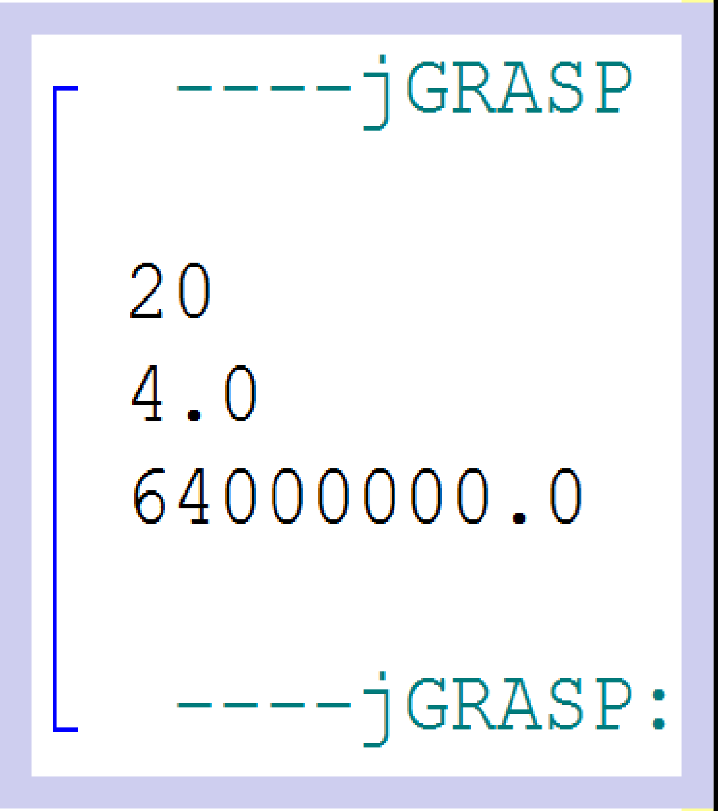


```python
# MathOperations05.py
# Python follows "Order of Operations"
# a.k.a. "PEMDAS"


print()

print (2 + 3 * 4)

print (14 - 6 / 2)

print (100 / 5 ** 2 * 3)

```

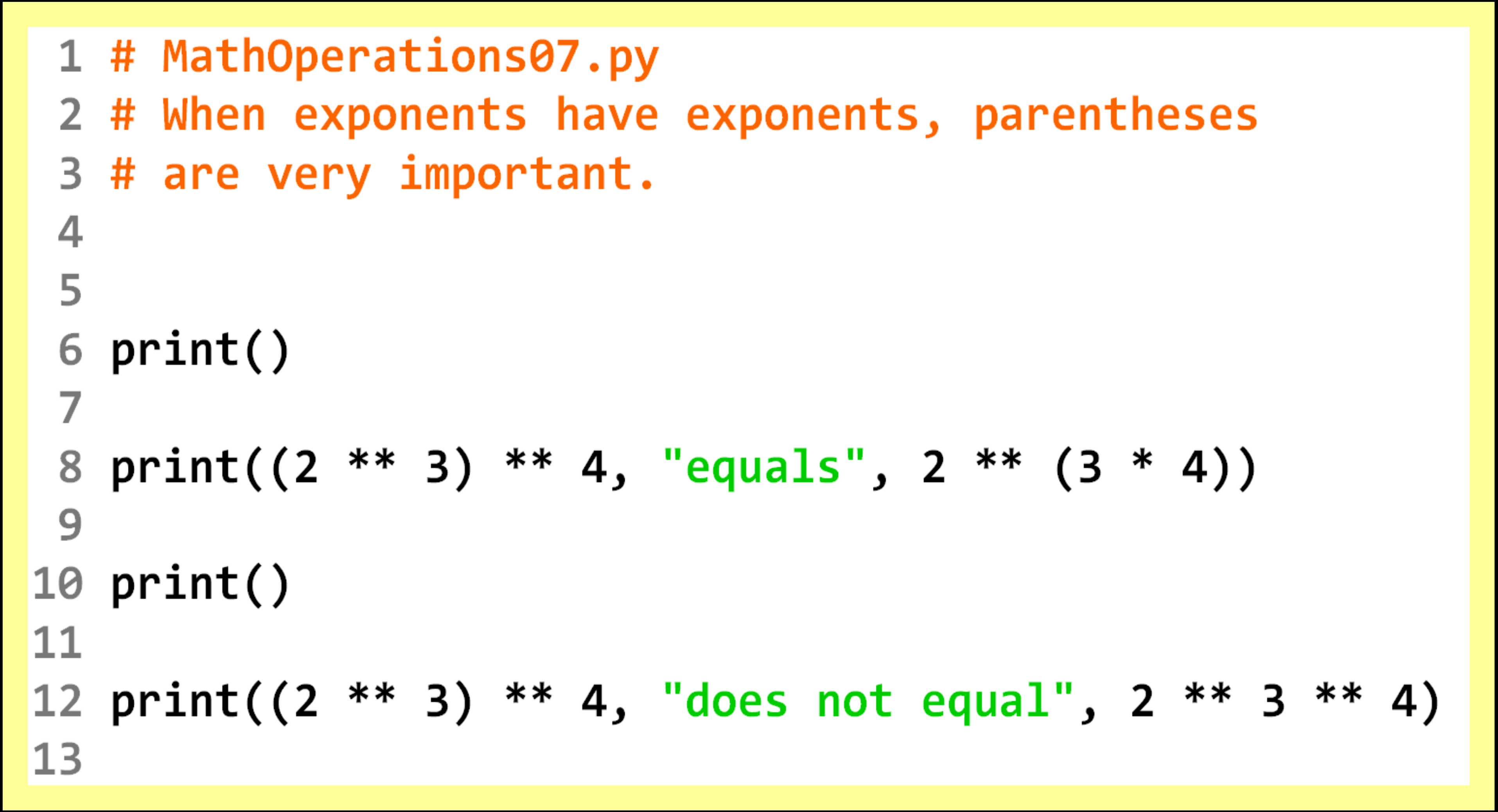
```
 ----jGRASP


14
11.0
12.0


 ----jGRASP:
```

```
# MathOperations06.py
# Parentheses make a difference.


print()

print ((2 + 3)  * 4)

print ((14 - 6) / 2)

print ((100 / 5) ** (2 * 3))

```

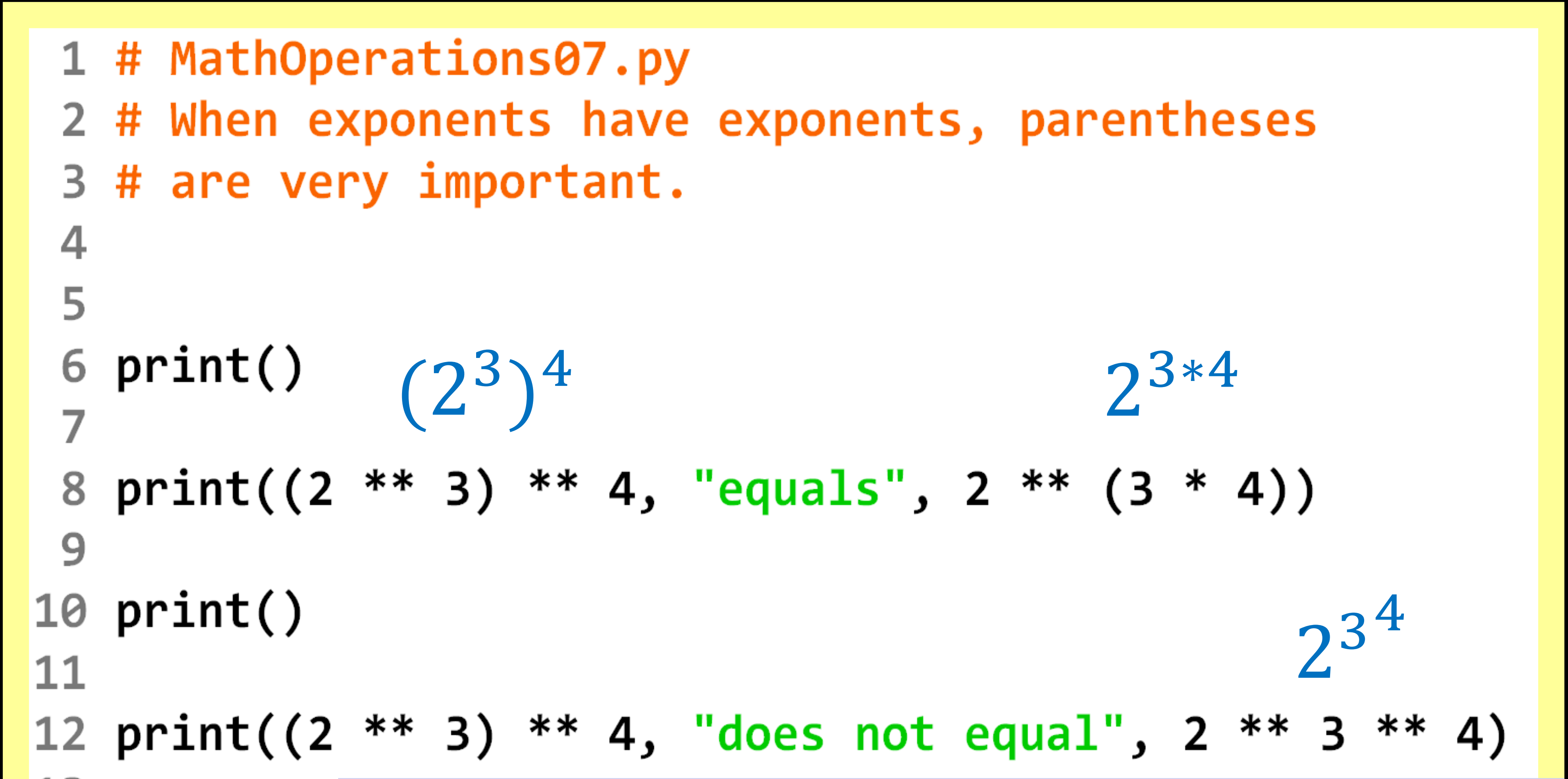
```python
# MathOperations06.py
# Parentheses make a difference.


print()

print ((2 + 3) * 4)

print ((14 - 6) / 2)

print ((100 / 5) ** (2 * 3))

```



```
 ----jGRASP

20
4.0
64000000.0

 ----jGRASP:
```



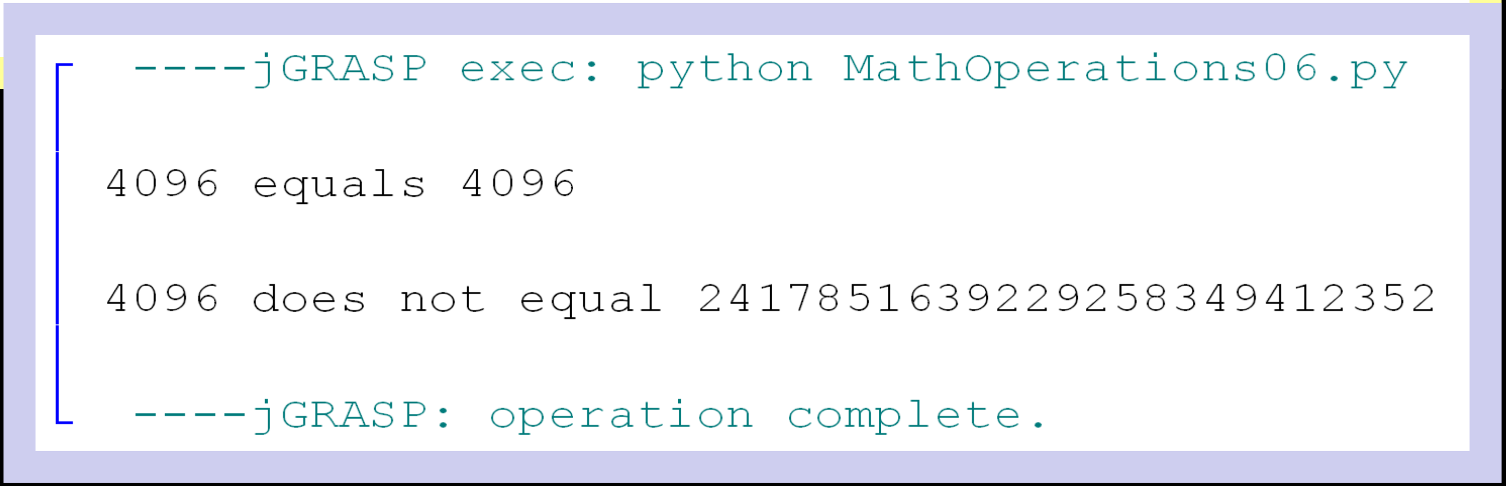
```python
# MathOperations07.py
# When exponents have exponents, parentheses
# are very important.


print()

print((2 ** 3) ** 4, "equals", 2 ** (3 * 4))

print()

print((2 ** 3) ** 4, "does not equal", 2 ** 3 ** 4)

```



```python
# MathOperations07.py
# When exponents have exponents, parentheses
# are very important.


print()

print((2 ** 3) ** 4, "equals", 2 ** (3 * 4))

print()

print((2 ** 3) ** 4, "does not equal", 2 ** 3 ** 4)

```

$(2^3)^4$ 2^{3*4}

2^{3^4}



```
 ----jGRASP exec: python MathOperations06.py

4096 equals 4096

4096 does not equal 2417851639229258349412352

 ----jGRASP: operation complete.
```

Section 4.3

Numeric Data Types

Variables

For centuries mathematicians and scientists have solved problems with equations and formulas. Computers are designed to solve those very same problems, but with much greater speed and accuracy.

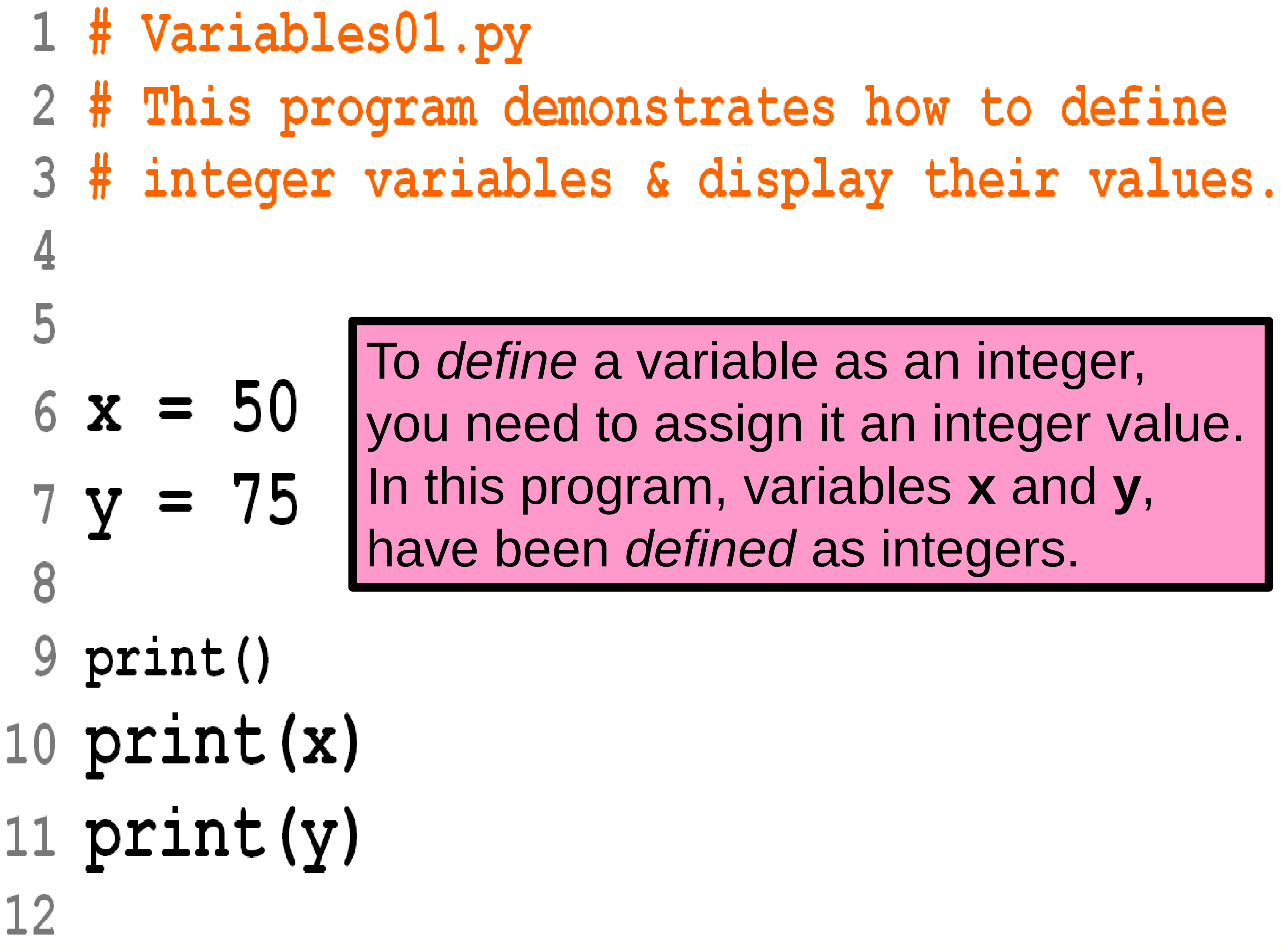
Equations and formulas use *variables*:

This Distance Formula

$$d = r * t$$

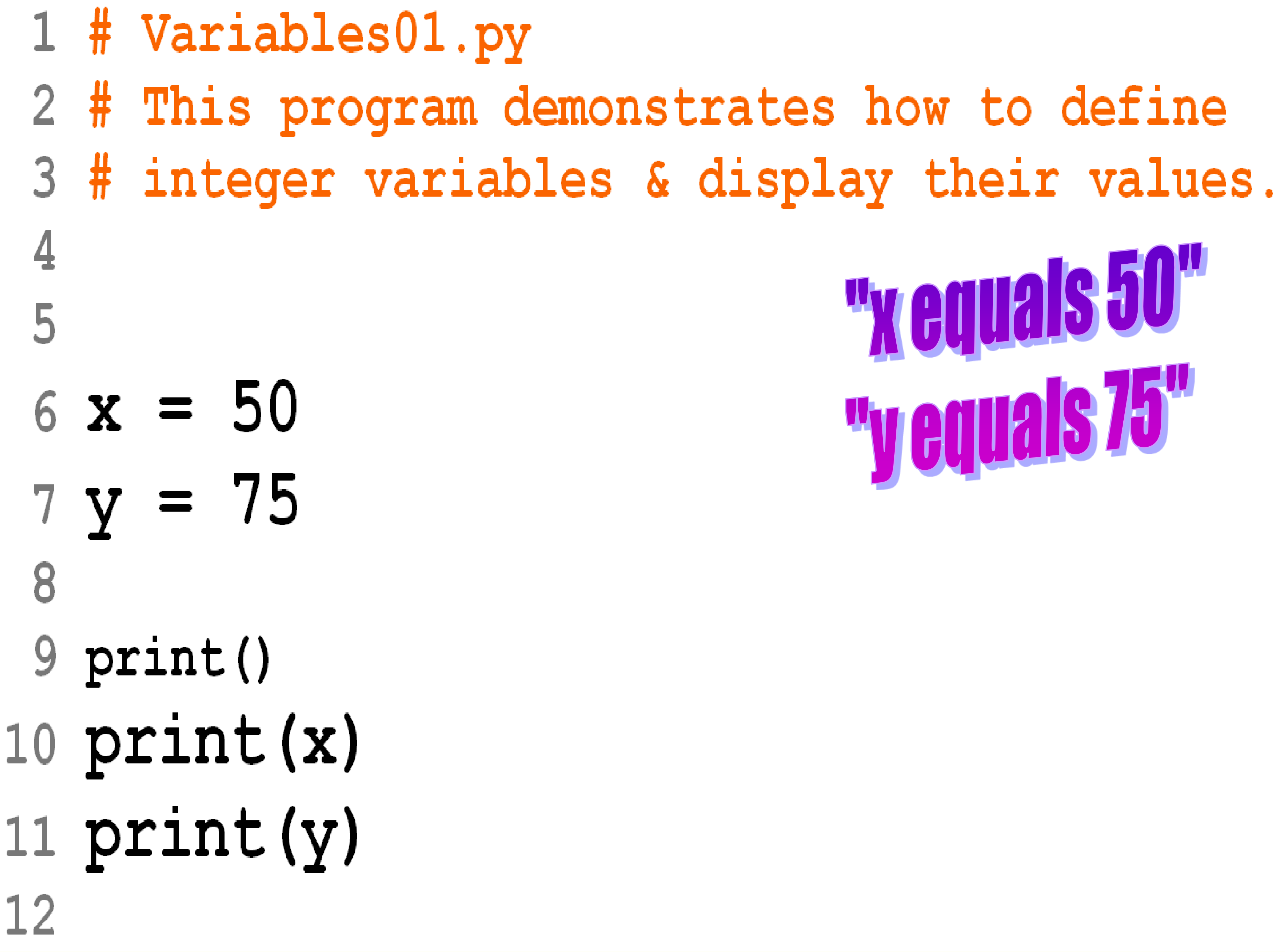
Convert Fahrenheit to Celsius

$$c = (f - 32) * 5/9$$



```
# Variables01.py
# This program demonstrates how to define
# integer variables & display their values.


x = 50
y = 75

print()
print(x)
print(y)

```

To *define* a variable as an integer, you need to assign it an integer value. In this program, variables **x** and **y**, have been *defined* as integers.



```
# Variables01.py
# This program demonstrates how to define
# integer variables & display their values.


x = 50
y = 75

print()
print(x)
print(y)

```

"x equals 50"

"y equals 75"

```
# Variables01.py
# This program demonstrates how to define
# integer variables & display their values.


x = 50
y = 75

print()
print(x)
print(y)

```

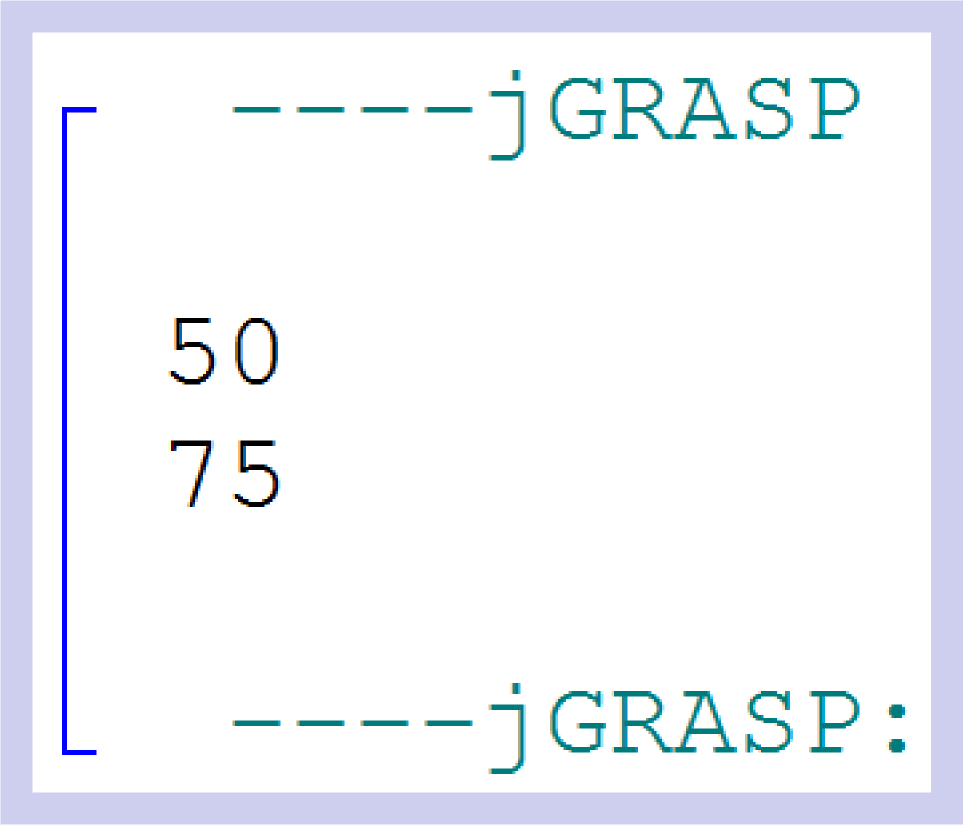
"x becomes 50"

"y becomes 75"

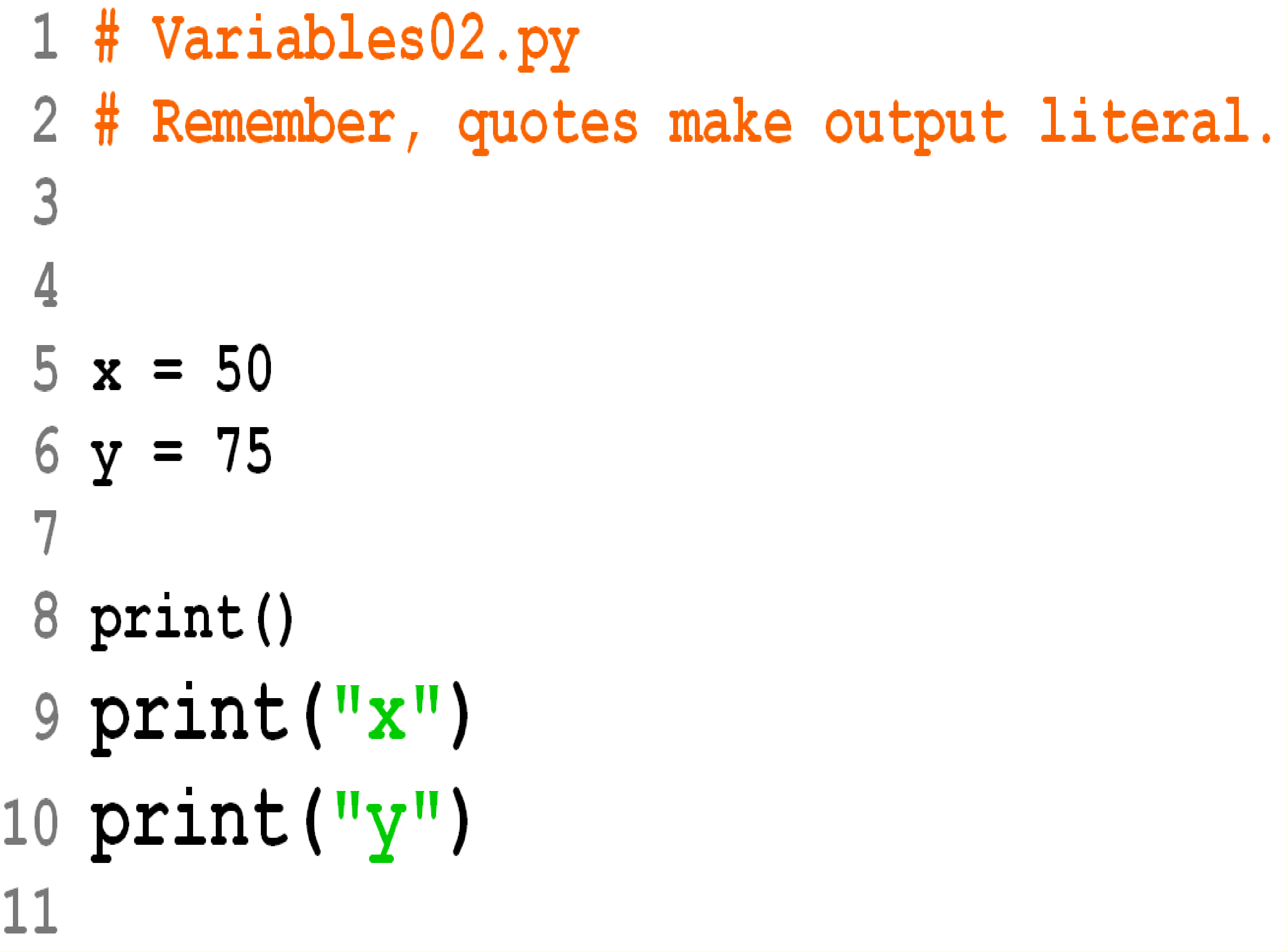
"x equals 50"
"y equals 75"

x	y
50	75

```python
# Variables01.py
# This program demonstrates how to define
# integer variables & display their values.


x = 50
y = 75

print()
print(x)
print(y)

```

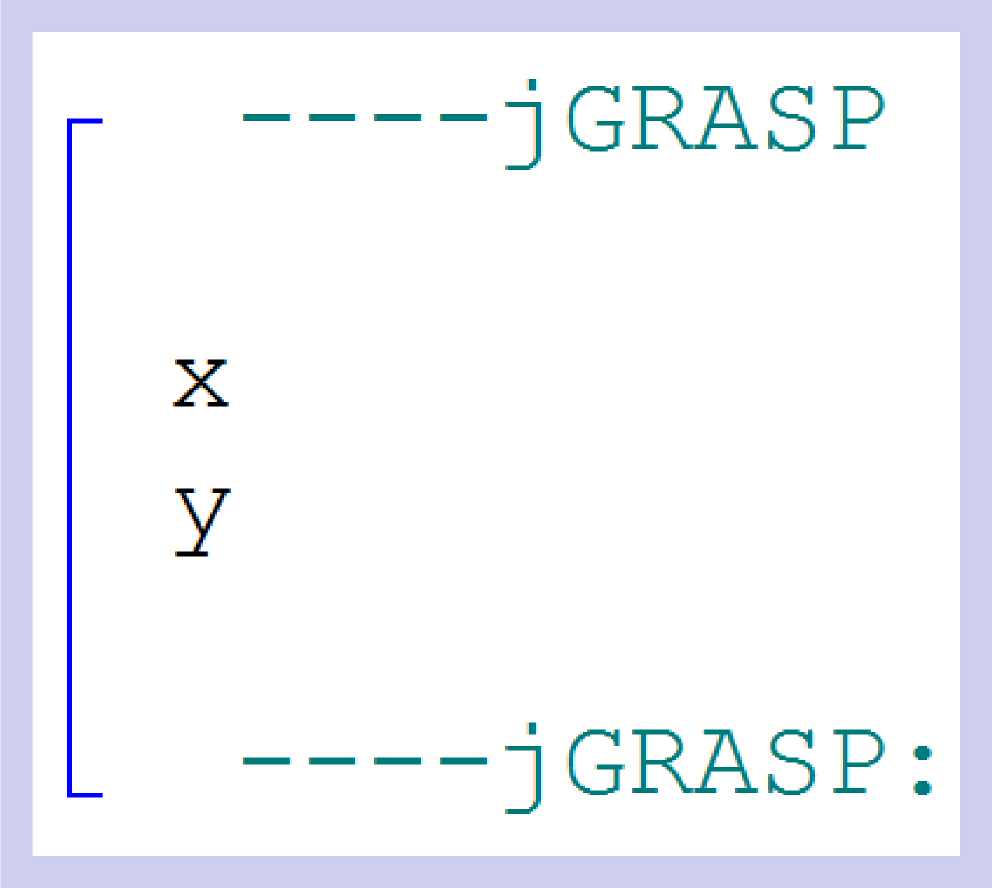


```
 ----jGRASP

50
75

 ----jGRASP:
```



```python
# Variables02.py
# Remember, quotes make output literal.


x = 50
y = 75

print()
print("x")
print("y")

```

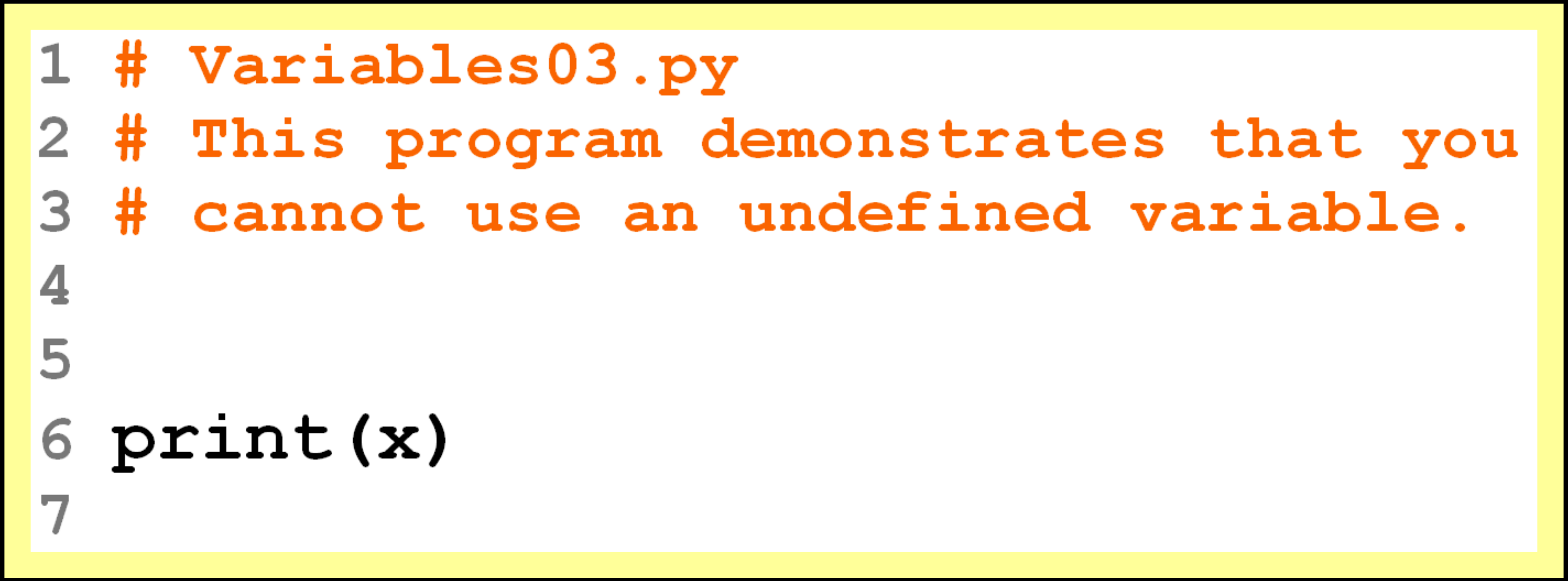
```python
# Variables02.py
# Remember, quotes make output literal.


x = 50
y = 75

print()
print("x")
print("y")

```



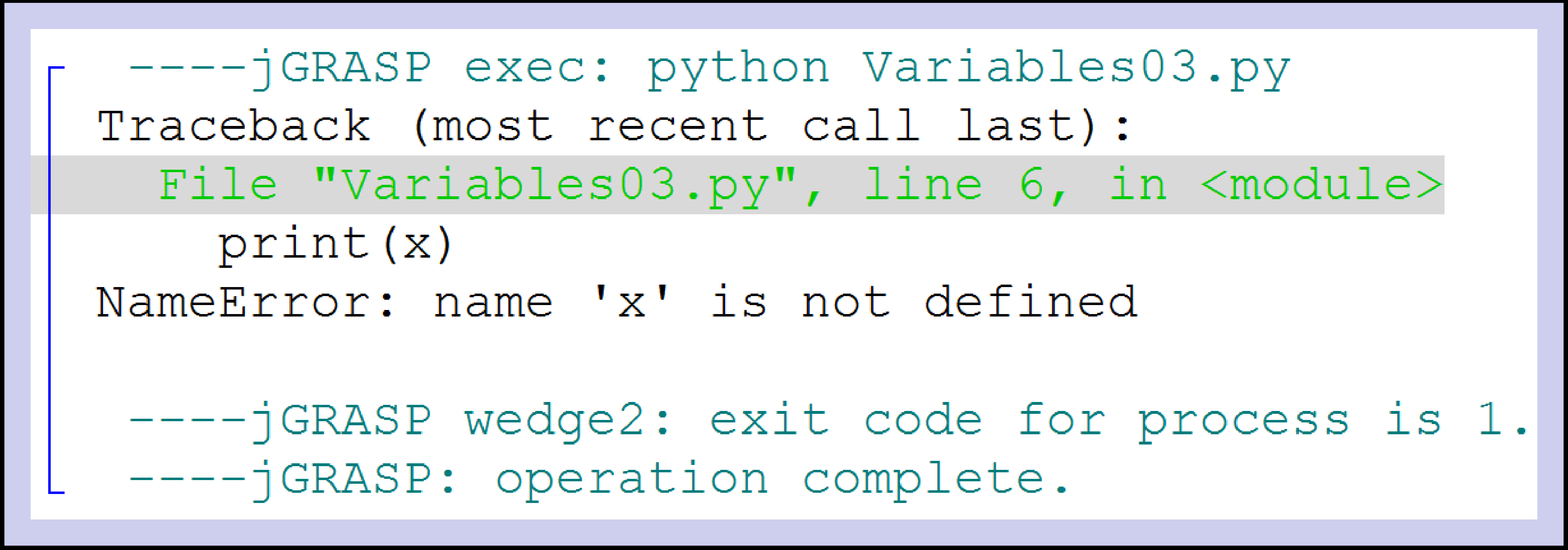
```
 ----jGRASP

x
y

 ----jGRASP:
```

print(x) vs. print("x")

print(x)	Display the value of the variable **x**.
print("x")	Display the letter "**x**".



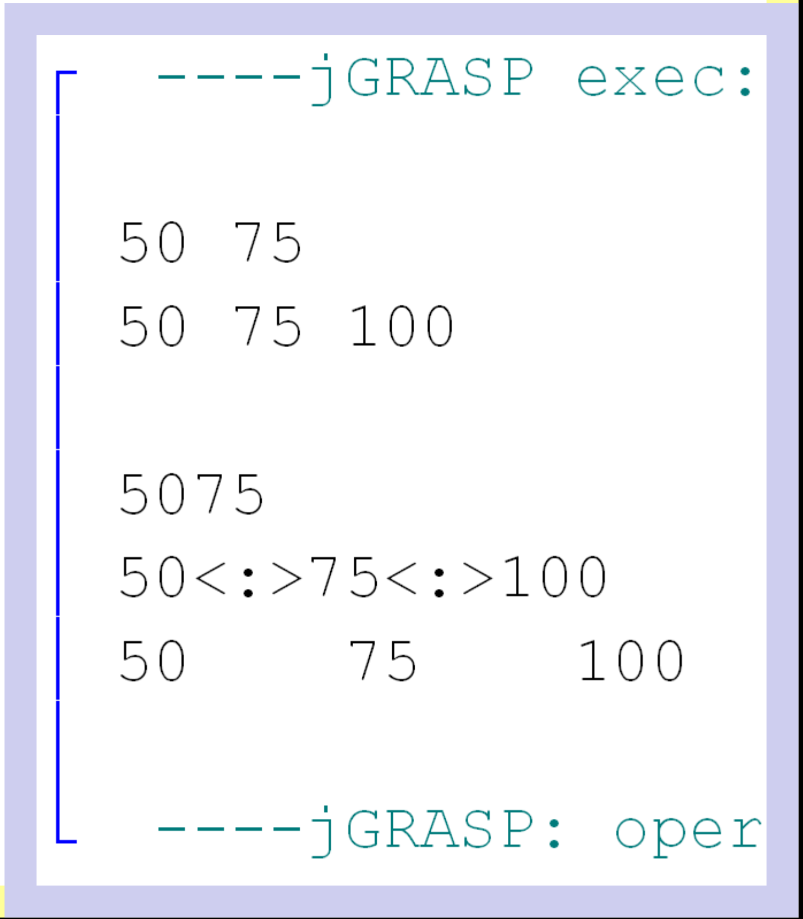
```python
# Variables03.py
# This program demonstrates that you
# cannot use an undefined variable.


print(x)

```



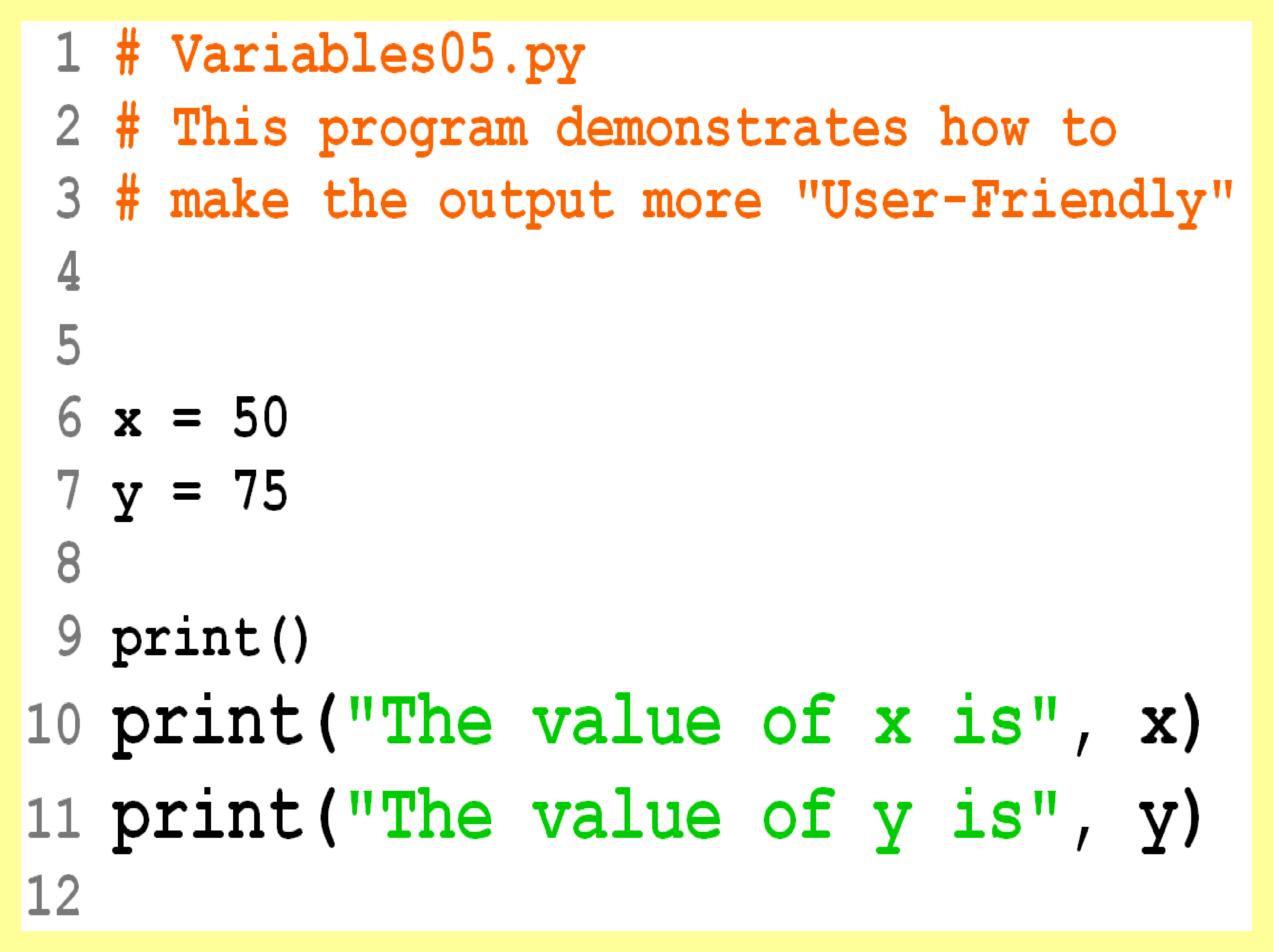
```
 ----jGRASP exec: python Variables03.py
Traceback (most recent call last):
  File "Variables03.py", line 6, in <module>
    print(x)
NameError: name 'x' is not defined

 ----jGRASP wedge2: exit code for process is 1.
 ----jGRASP: operation complete.
```

```python
# Variables04.py
# This program demonstrates how to print the
# values of multiple variables on one line.
# By default, these values are separated by
# a single space, but this can be changed
# by using <sep> at the end of <print>.


x = 50
y = 75
z = 100

print()
print(x,y)
print(x,y,z)
print()
print(x,y,sep="")
print(x,y,z,sep="<:>")
print(x,y,z,sep="\t\t")
```

```python
# Variables04.py
# This program demonstrates how to print the
# values of multiple variables on one line.
# By default, these values are separated by
# a single space, but this can be changed
# by using <sep> at the end of <print>.


x = 50
y = 75
z = 100

print()
print(x,y)
print(x,y,z)
print()
print(x,y,sep="")
print(x,y,z,sep="<:>")
print(x,y,z,sep="\t\t")
```

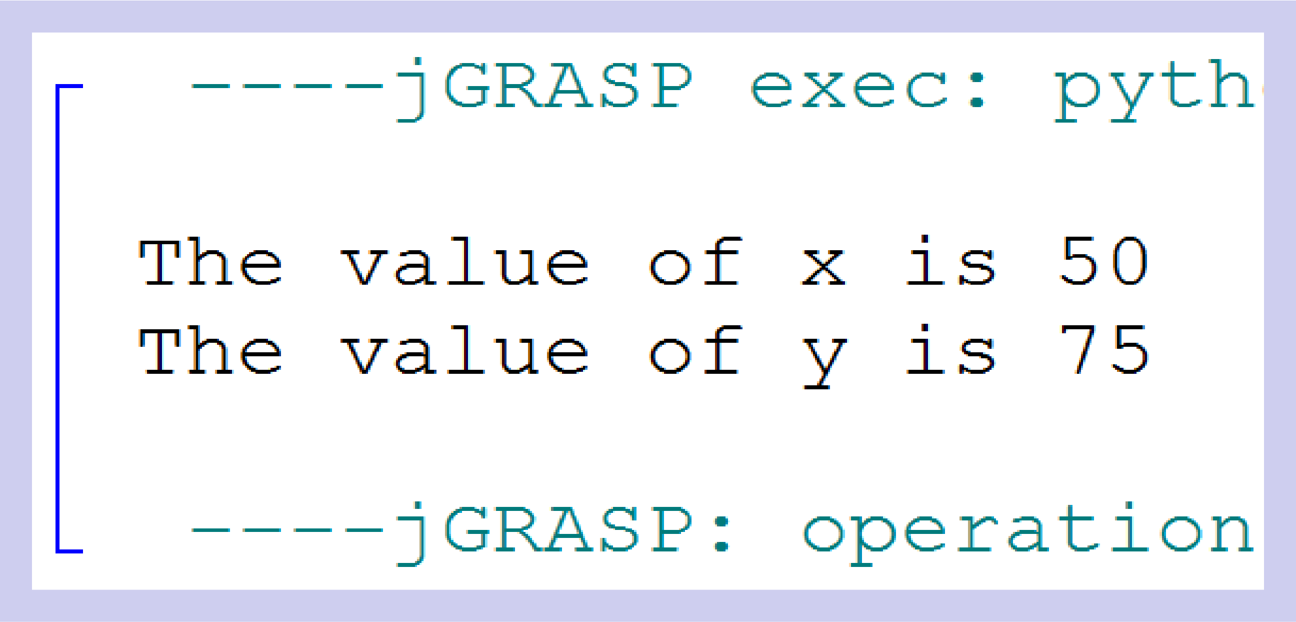


```
 ----jGRASP exec:

50 75
50 75 100

5075
50<:>75<:>100
50      75      100

 ----jGRASP: oper
```



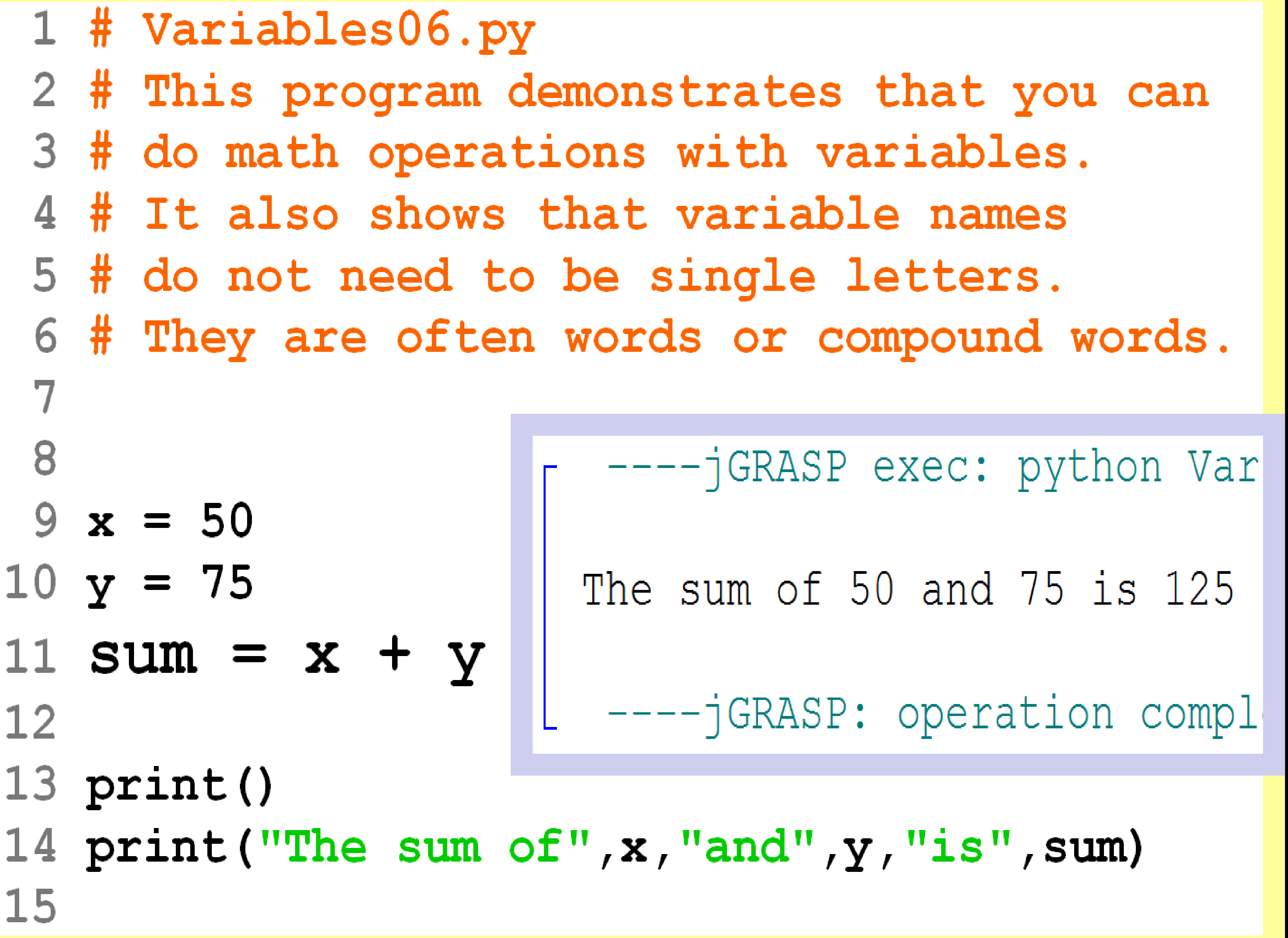
```
# Variables05.py
# This program demonstrates how to
# make the output more "User-Friendly"


x = 50
y = 75

print()
print("The value of x is", x)
print("The value of y is", y)

```

```python
# Variables05.py
# This program demonstrates how to
# make the output more "User-Friendly"


x = 50
y = 75

print()
print("The value of x is", x)
print("The value of y is", y)

```



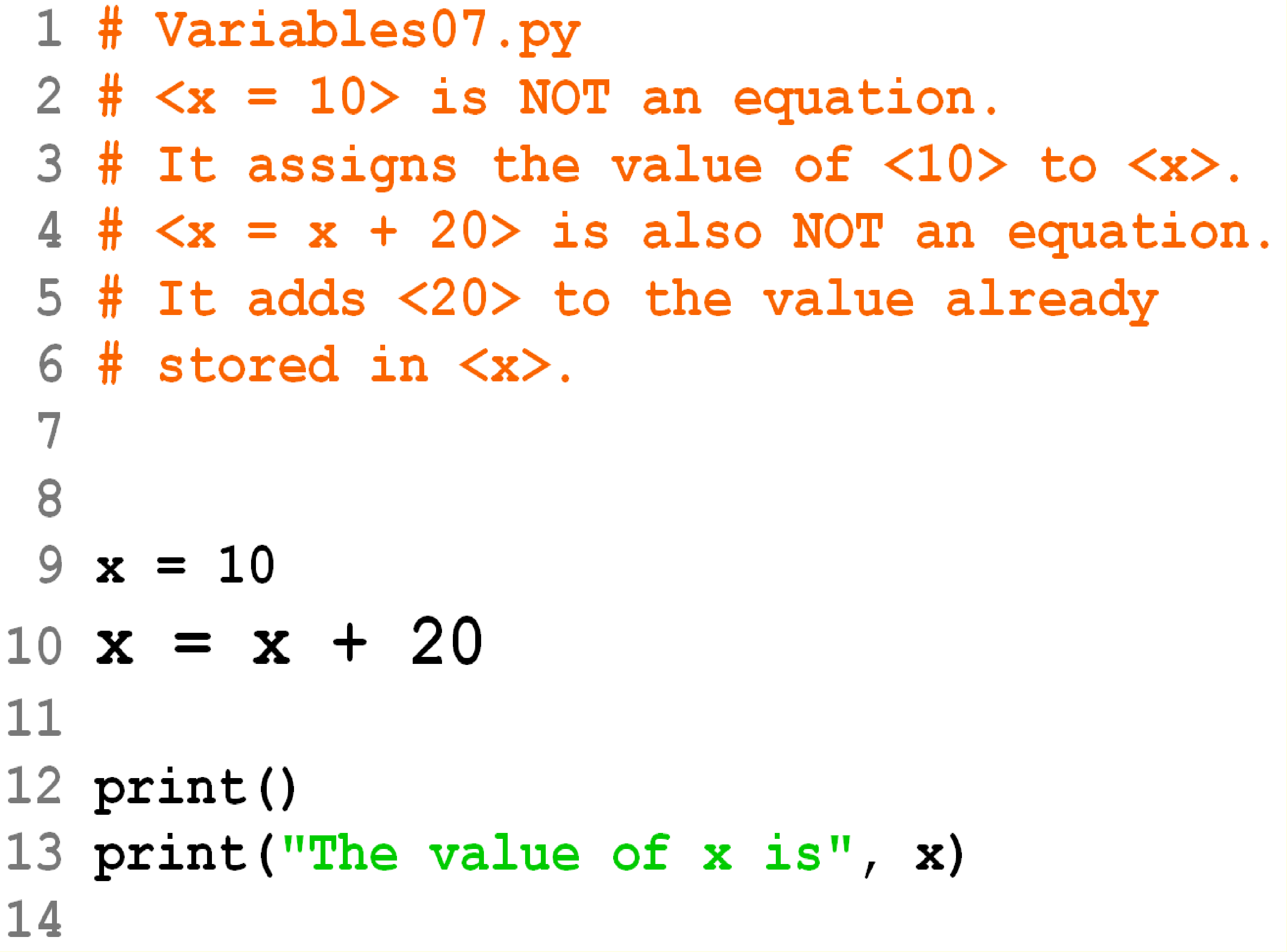
```
 ----jGRASP exec: pyth

The value of x is 50
The value of y is 75


 ----jGRASP: operation
```

```python
# Variables06.py
# This program demonstrates that you can
# do math operations with variables.
# It also shows that variable names
# do not need to be single letters.
# They are often words or compound words.


x = 50
y = 75
sum = x + y

print()
print("The sum of",x,"and",y,"is",sum)

```



```python
# Variables06.py
# This program demonstrates that you can
# do math operations with variables.
# It also shows that variable names
# do not need to be single letters.
# They are often words or compound words.


x = 50
y = 75
sum = x + y

print()
print("The sum of",x,"and",y,"is",sum)

```

```
 ----jGRASP exec: python Var

The sum of 50 and 75 is 125

 ----jGRASP: operation compl
```



```python
# Variables07.py
# <x = 10> is NOT an equation.
# It assigns the value of <10> to <x>.
# <x = x + 20> is also NOT an equation.
# It adds <20> to the value already
# stored in <x>.


x = 10
x = x + 20

print()
print("The value of x is", x)
```

How can x = x + 20?

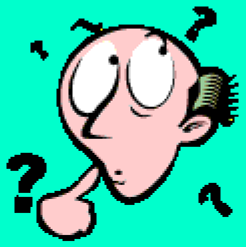
If you are thinking of **x = x + 20** as being an equation – it will not work:

$$x = x + 20$$

$$\underline{-x \quad -x}$$

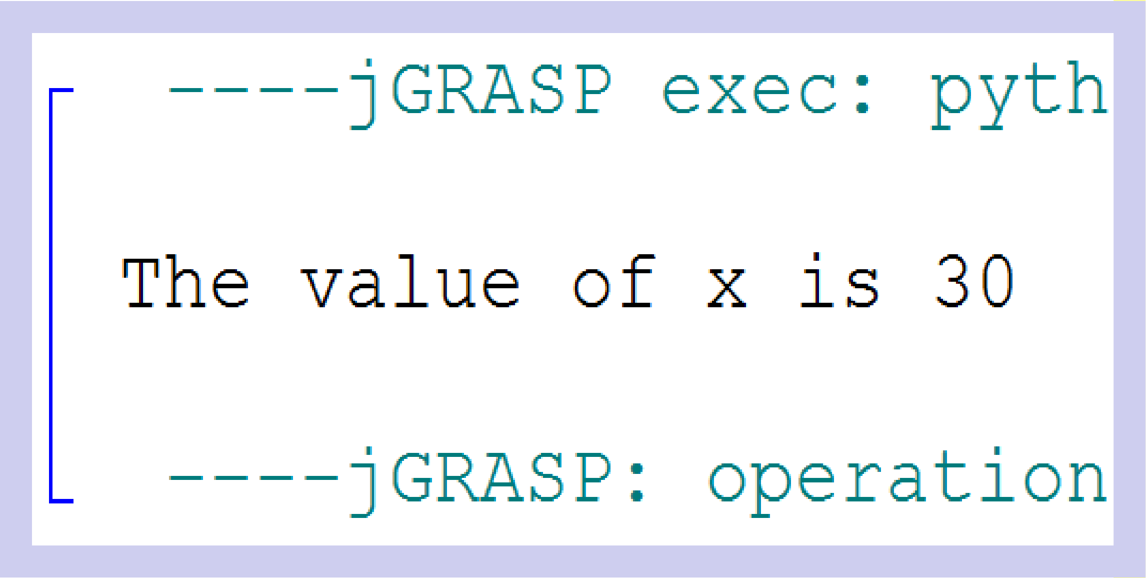
$$0 = 0 + 20$$

$$\mathbf{0 = 20}$$



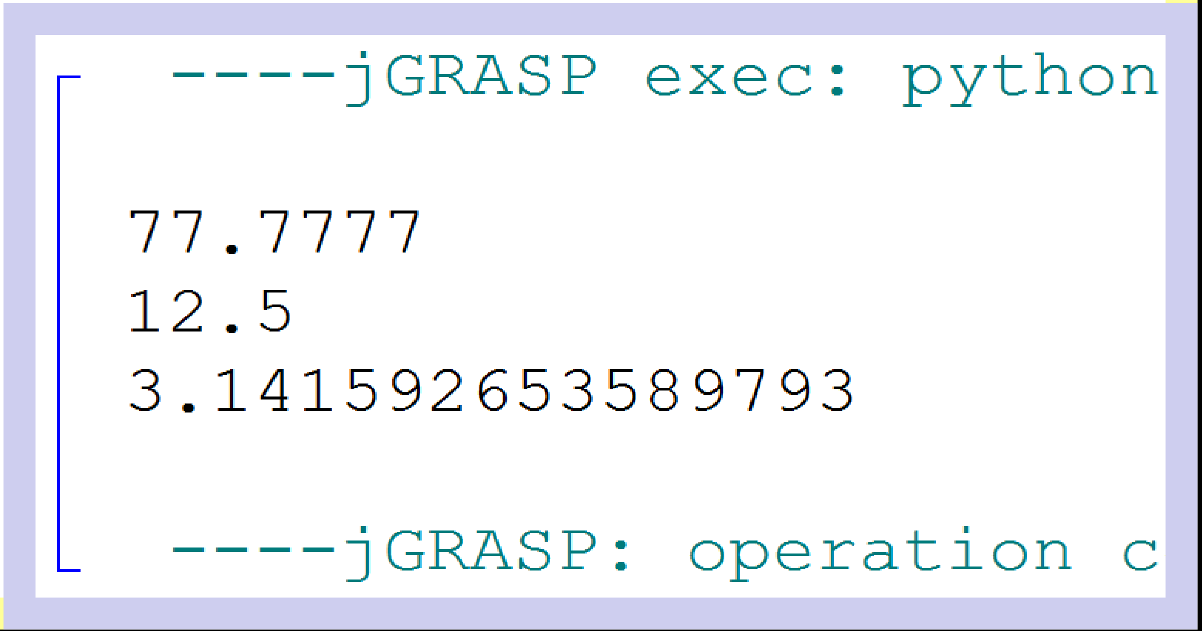
Instead, you need to be thinking that the *new value* of x will <u>become</u> 20 more than the *old value* of x.

```
# Variables07.py
# <x = 10> is NOT an equation.
# It assigns the value of <10> to <x>.
# <x = x + 20> is also NOT an equation.
# It adds <20> to the value already
# stored in <x>.


x = 10
x = x + 20

print()
print("The value of x is", x)

```



```
 ----jGRASP exec: pyth

The value of x is 30

 ----jGRASP: operation
```

```python
# Variables08.py
# The program shows that variables can
# also store real number values.


q = 77.7777
r = 12.5
pi = 3.141592653589793

print()
print(q)
print(r)
print(pi)

```

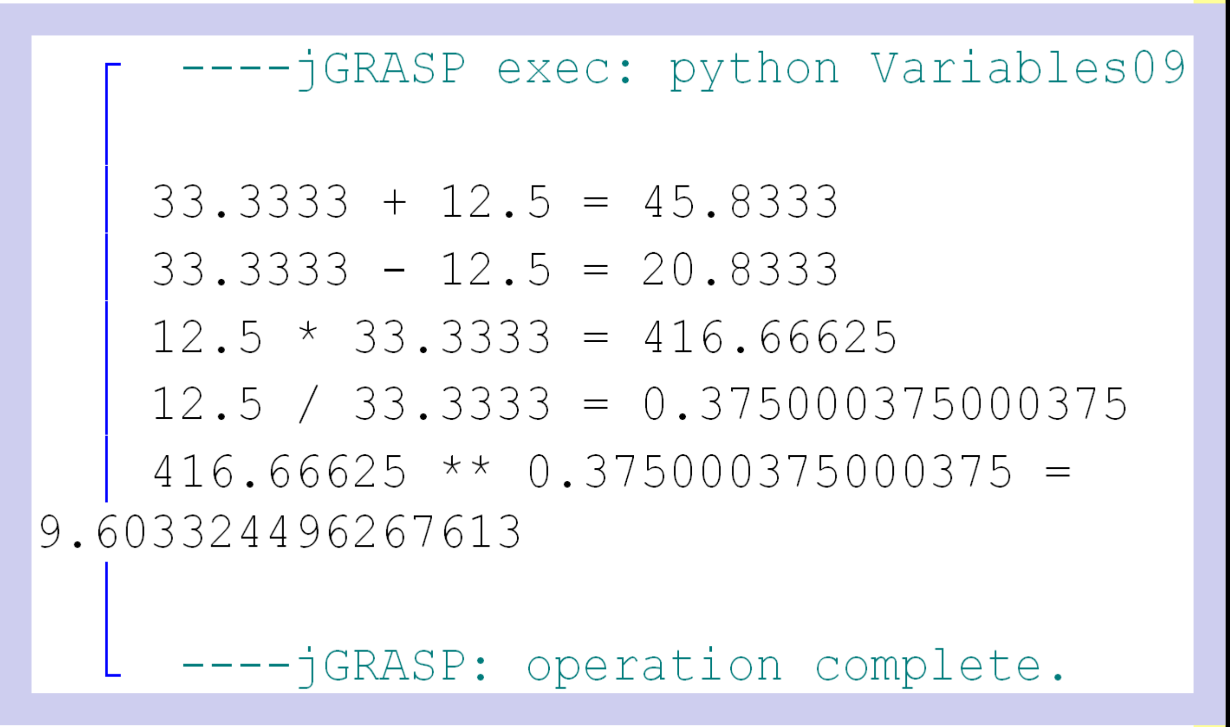


```
 ----jGRASP exec: python

77.7777
12.5
3.141592653589793

 ----jGRASP: operation c
```

```python
# Variables09.py
# The program demonstrates that the same
# mathematical operations that work with
# integers, also work with real numbers.
# NOTE: While "integer division" and
# "remainder division" operators technically
# work with real numbers, they are not
# really meant for them and should only
# be used with integers.


q = 33.3333
r = 12.5

```

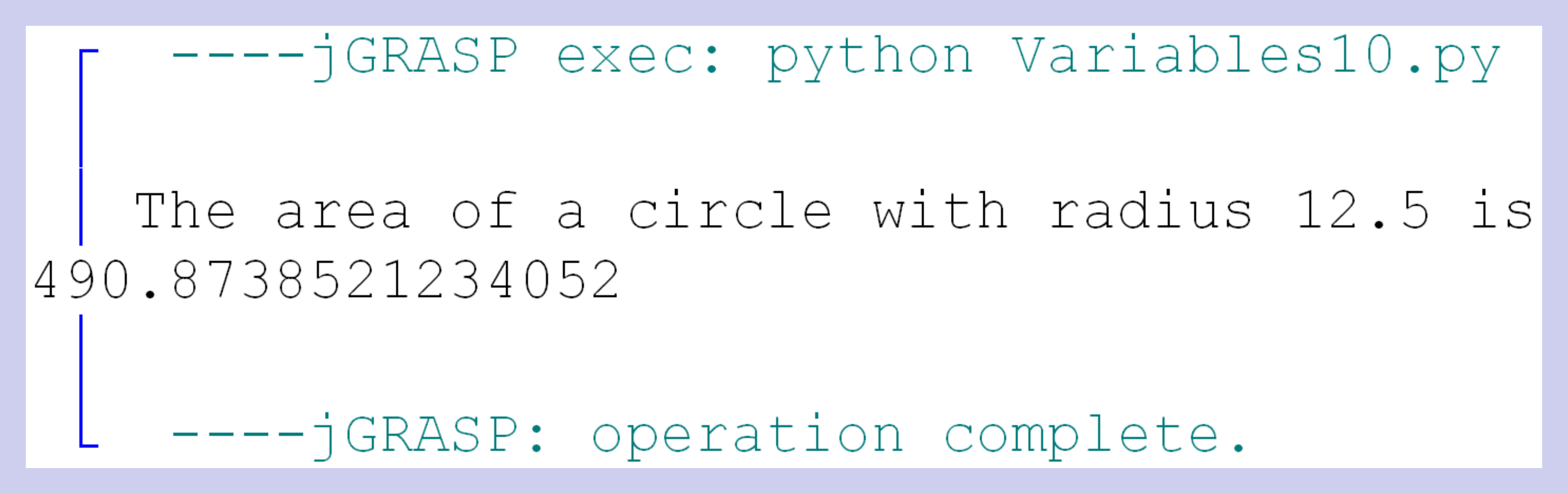
```
q = 33.3333
r = 12.5

a = q + r
b = q - r
c = r * q
d = r / q
e = c ** d

print()
print(q,"+",r,"=",a)
print(q,"-",r,"=",b)
print(r,"*",q,"=",c)
print(r,"/",q,"=",d)
print(c,"**",d,"=",e)
```

```python
q = 33.3333
r = 12.5

a = q + r
b = q - r
c = r * q
d = r / q
e = c ** d

print()
print(q,"+",r,"=",a)
print(q,"-",r,"=",b)
print(r,"*",q,"=",c)
print(r,"/",q,"=",d)
print(c,"**",d,"=",e)
```

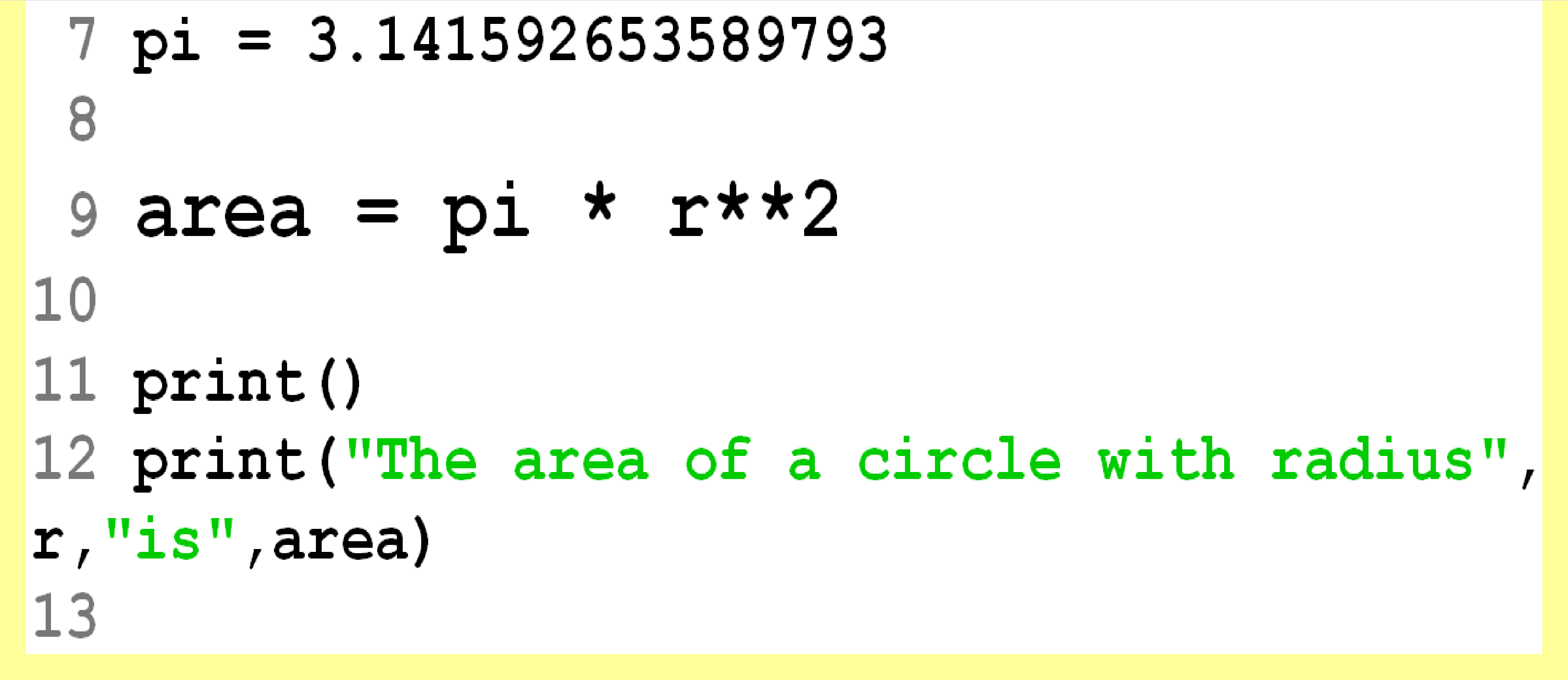


```
 ----jGRASP exec: python Variables09

 33.3333 + 12.5 = 45.8333
 33.3333 - 12.5 = 20.8333
 12.5 * 33.3333 = 416.66625
 12.5 / 33.3333 = 0.375000375000375
 416.66625 ** 0.375000375000375 =
9.603324496267613

 ----jGRASP: operation complete.
```

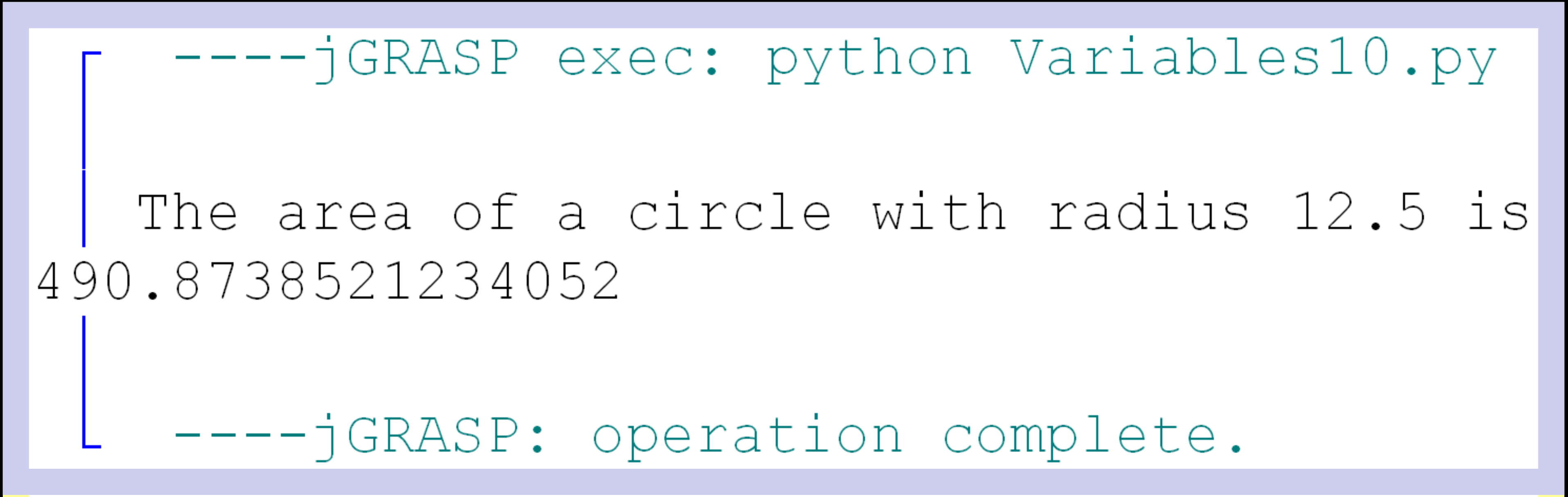
```python
# Variables10.py
# The program demonstrates a
# practical use of variables.


r = 12.5
pi = 3.141592653589793

area = pi * r**2

print()
print("The area of a circle with radius",r,"is",area)

```



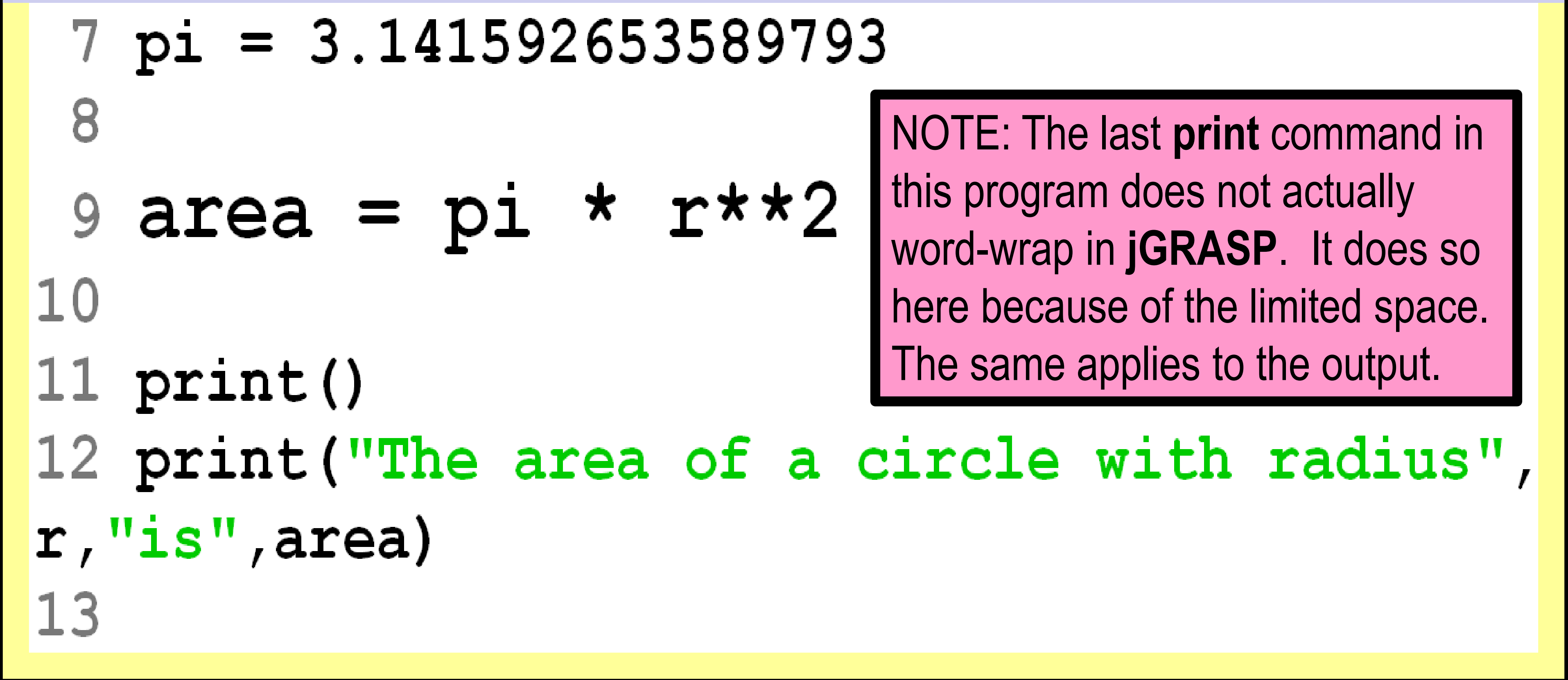
```
 ----jGRASP exec: python Variables10.py

The area of a circle with radius 12.5 is
490.8738521234052

 ----jGRASP: operation complete.
```



```
pi = 3.141592653589793

area = pi * r**2

print()
print("The area of a circle with radius",
r,"is",area)

```



```
 ----jGRASP exec: python Variables10.py

The area of a circle with radius 12.5 is
490.8738521234052

 ----jGRASP: operation complete.
```



```
pi = 3.141592653589793

area = pi * r**2

print()
print("The area of a circle with radius",
r,"is",area)

```

NOTE: The last **print** command in this program does not actually word-wrap in **jGRASP**. It does so here because of the limited space. The same applies to the output.

Beware of Hidden Operators in Mathematics

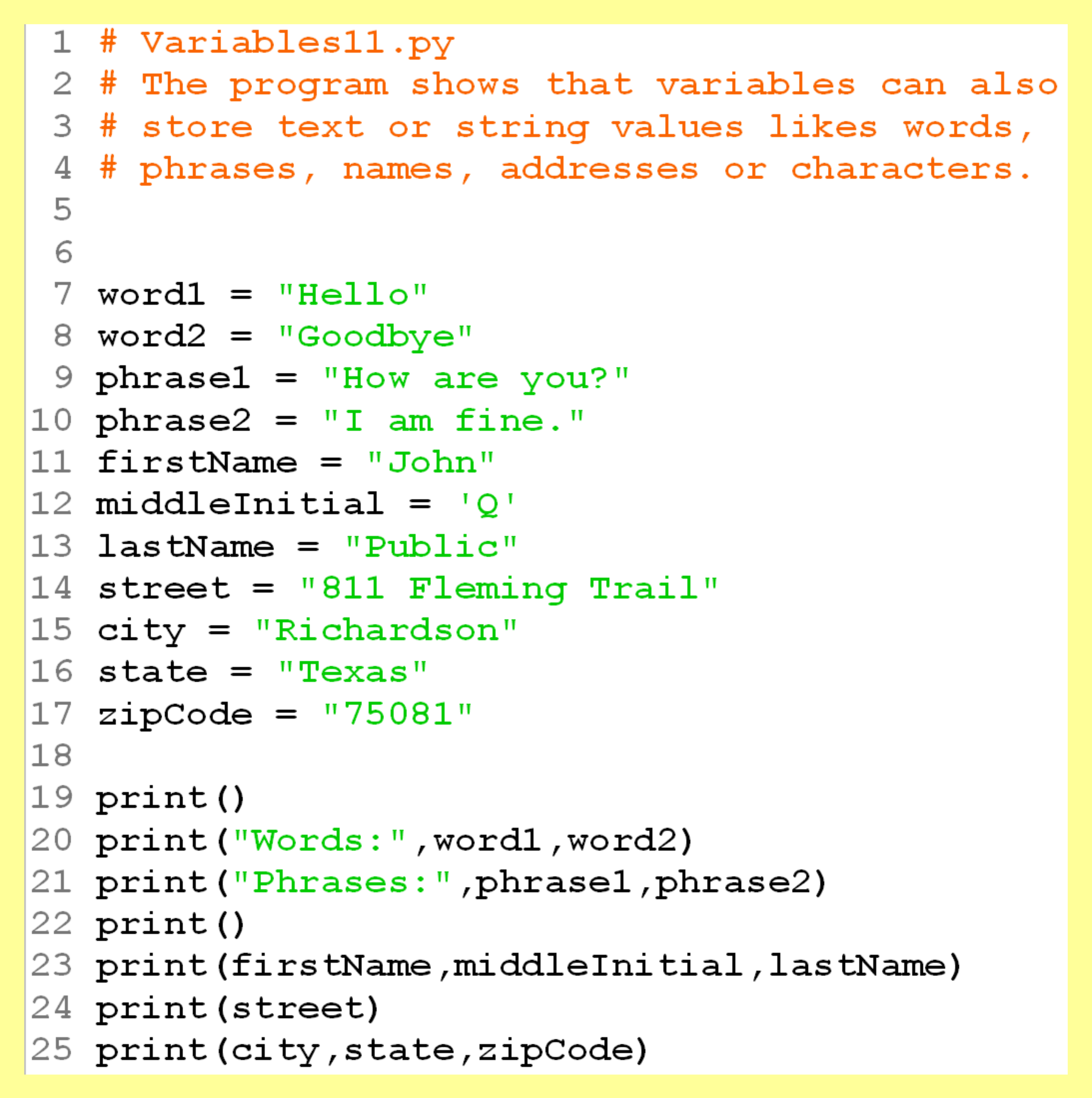
Mathematics	Python Source Code
$5AB$	5 * A * B
$\frac{5}{7}$	5 / 7
$\frac{A + B}{A – B}$	A + B / A – B **What is wrong with this one?**

Beware of Hidden Operators in Mathematics

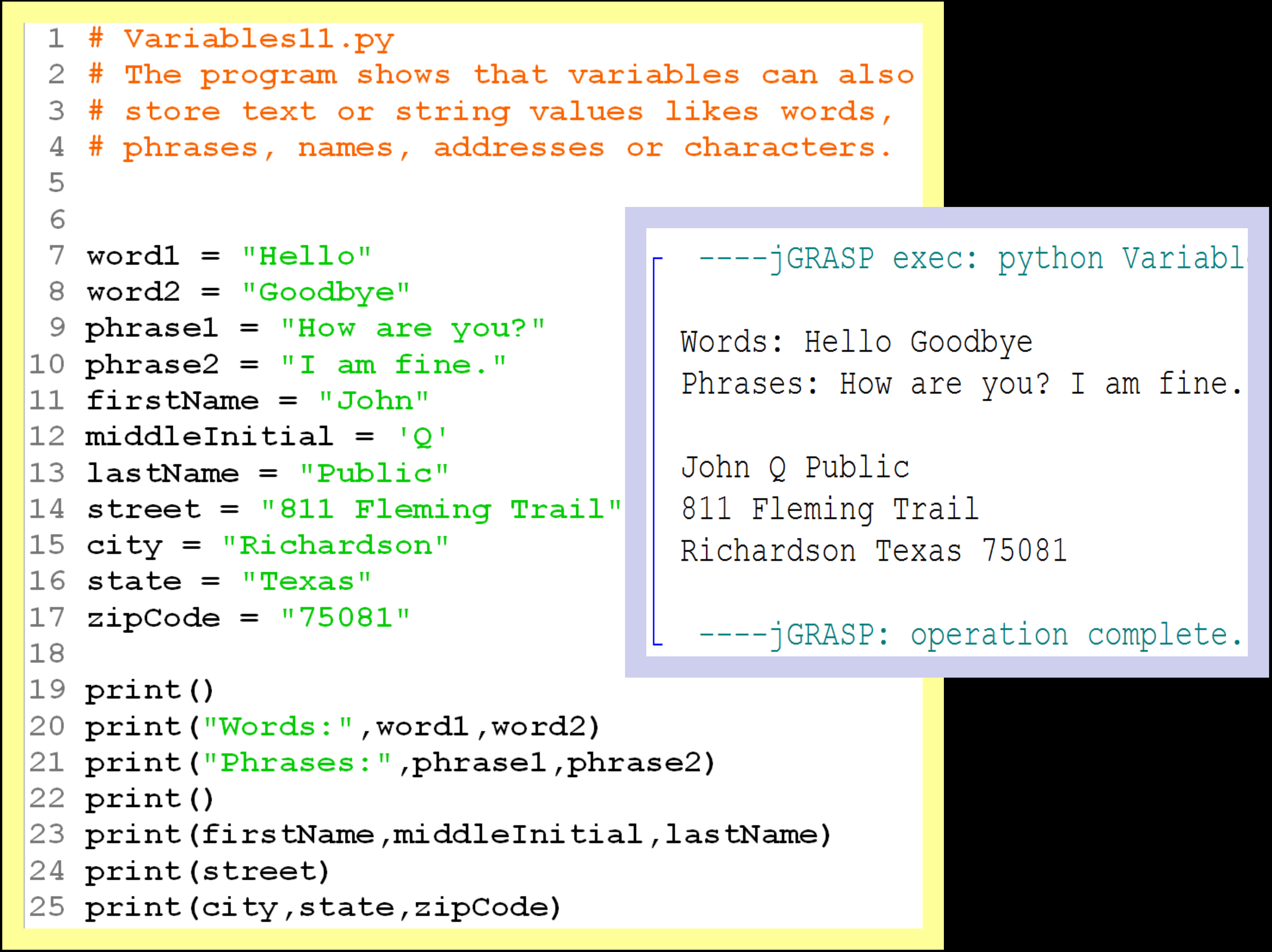
Mathematics	Python Source Code
5AB	`5 * A * B`
$\frac{5}{7}$	`5 / 7`
$\frac{A + B}{A - B}$	`(A + B) / (A – B)` **Always remember PEMDAS!**

Section 4.4

Non-Numeric Data Types



```python
# Variables11.py
# The program shows that variables can also
# store text or string values likes words,
# phrases, names, addresses or characters.


word1 = "Hello"
word2 = "Goodbye"
phrase1 = "How are you?"
phrase2 = "I am fine."
firstName = "John"
middleInitial = 'Q'
lastName = "Public"
street = "811 Fleming Trail"
city = "Richardson"
state = "Texas"
zipCode = "75081"

print()
print("Words:",word1,word2)
print("Phrases:",phrase1,phrase2)
print()
print(firstName,middleInitial,lastName)
print(street)
print(city,state,zipCode)
```

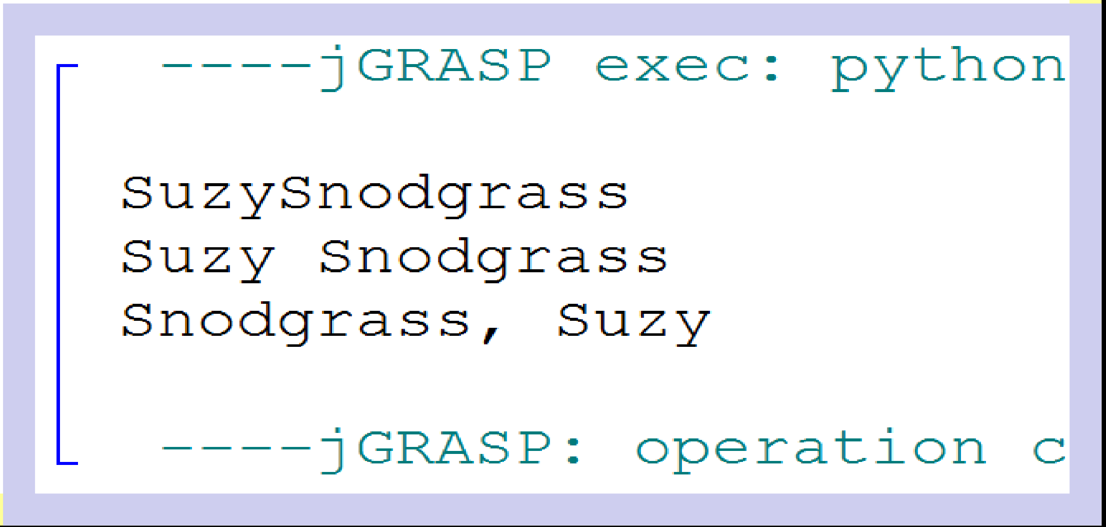


```
# Variables11.py
# The program shows that variables can also
# store text or string values likes words,
# phrases, names, addresses or characters.


word1 = "Hello"
word2 = "Goodbye"
phrase1 = "How are you?"
phrase2 = "I am fine."
firstName = "John"
middleInitial = 'Q'
lastName = "Public"
street = "811 Fleming Trail"
city = "Richardson"
state = "Texas"
zipCode = "75081"

print()
print("Words:",word1,word2)
print("Phrases:",phrase1,phrase2)
print()
print(firstName,middleInitial,lastName)
print(street)
print(city,state,zipCode)
```

```
 ----jGRASP exec: python Variabl

Words: Hello Goodbye
Phrases: How are you? I am fine.

John Q Public
811 Fleming Trail
Richardson Texas 75081

 ----jGRASP: operation complete.
```

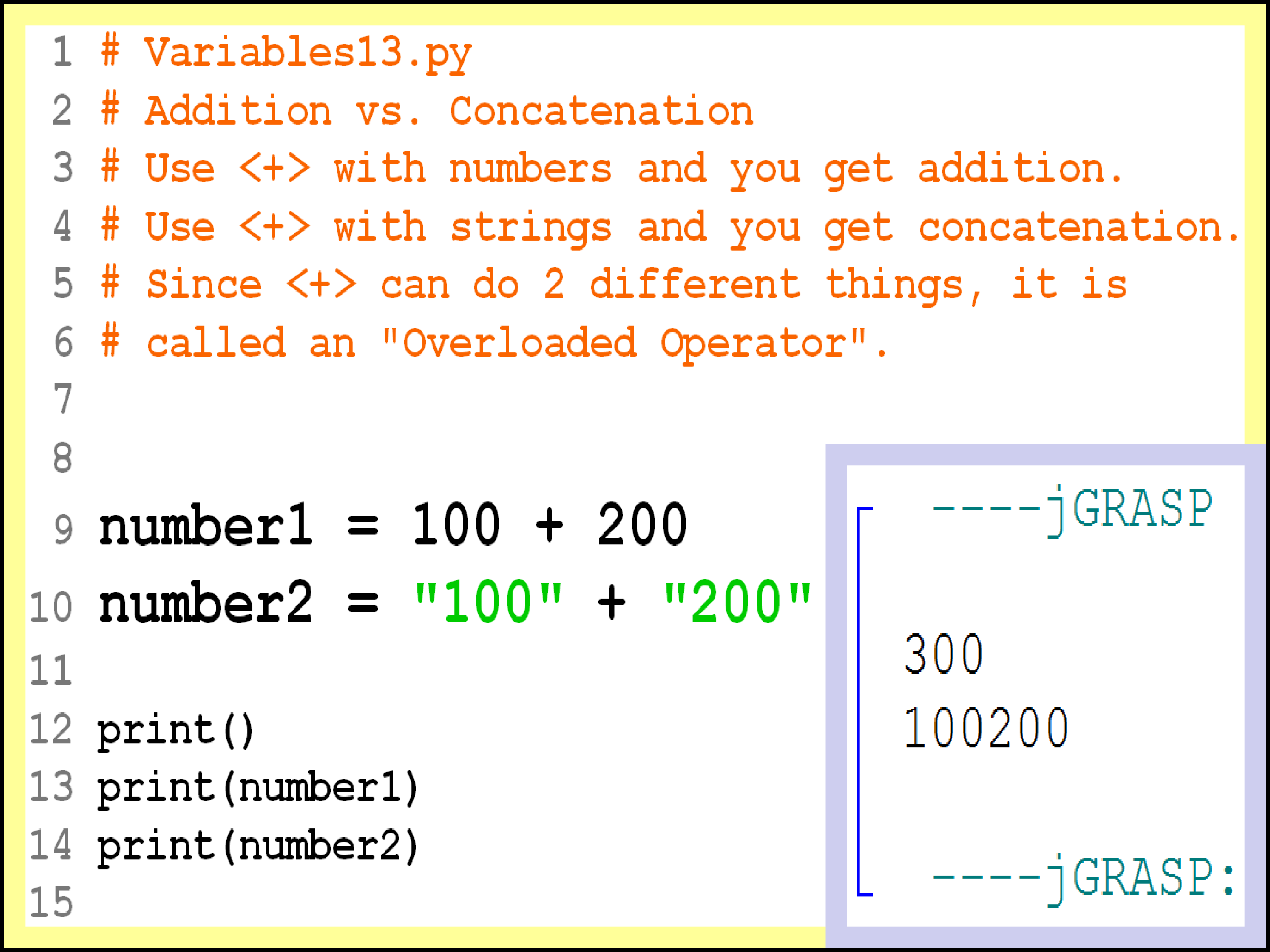
```python
# Variables12.py
# This program demonstrates "String Concatenation"
# which is joining together 2 or more strings.

firstName = "Suzy"
lastName = "Snodgrass"

fullName1 = firstName + lastName
fullName2 = firstName + " " + lastName
fullName3 = lastName + ", " + firstName

print()
print(fullName1)
print(fullName2)
print(fullName3)

```

```python
# Variables12.py
# This program demonstrates "String Concatenation"
# which is joining together 2 or more strings.

firstName = "Suzy"
lastName = "Snodgrass"

fullName1 = firstName + lastName
fullName2 = firstName + " " + lastName
fullName3 = lastName + ", " + firstName

print()
print(fullName1)
print(fullName2)
print(fullName3)
```



```
 ----jGRASP exec: python

SuzySnodgrass
Suzy Snodgrass
Snodgrass, Suzy

 ----jGRASP: operation c
```

```python
# Variables13.py
# Addition vs. Concatenation
# Use <+> with numbers and you get addition.
# Use <+> with strings and you get concatenation.
# Since <+> can do 2 different things, it is
# called an "Overloaded Operator".


number1 = 100 + 200
number2 = "100" + "200"

print()
print(number1)
print(number2)

```



```
# Variables13.py
# Addition vs. Concatenation
# Use <+> with numbers and you get addition.
# Use <+> with strings and you get concatenation.
# Since <+> can do 2 different things, it is
# called an "Overloaded Operator".


number1 = 100 + 200
number2 = "100" + "200"

print()
print(number1)
print(number2)

```

```
 ----jGRASP

300
100200

 ----jGRASP:
```

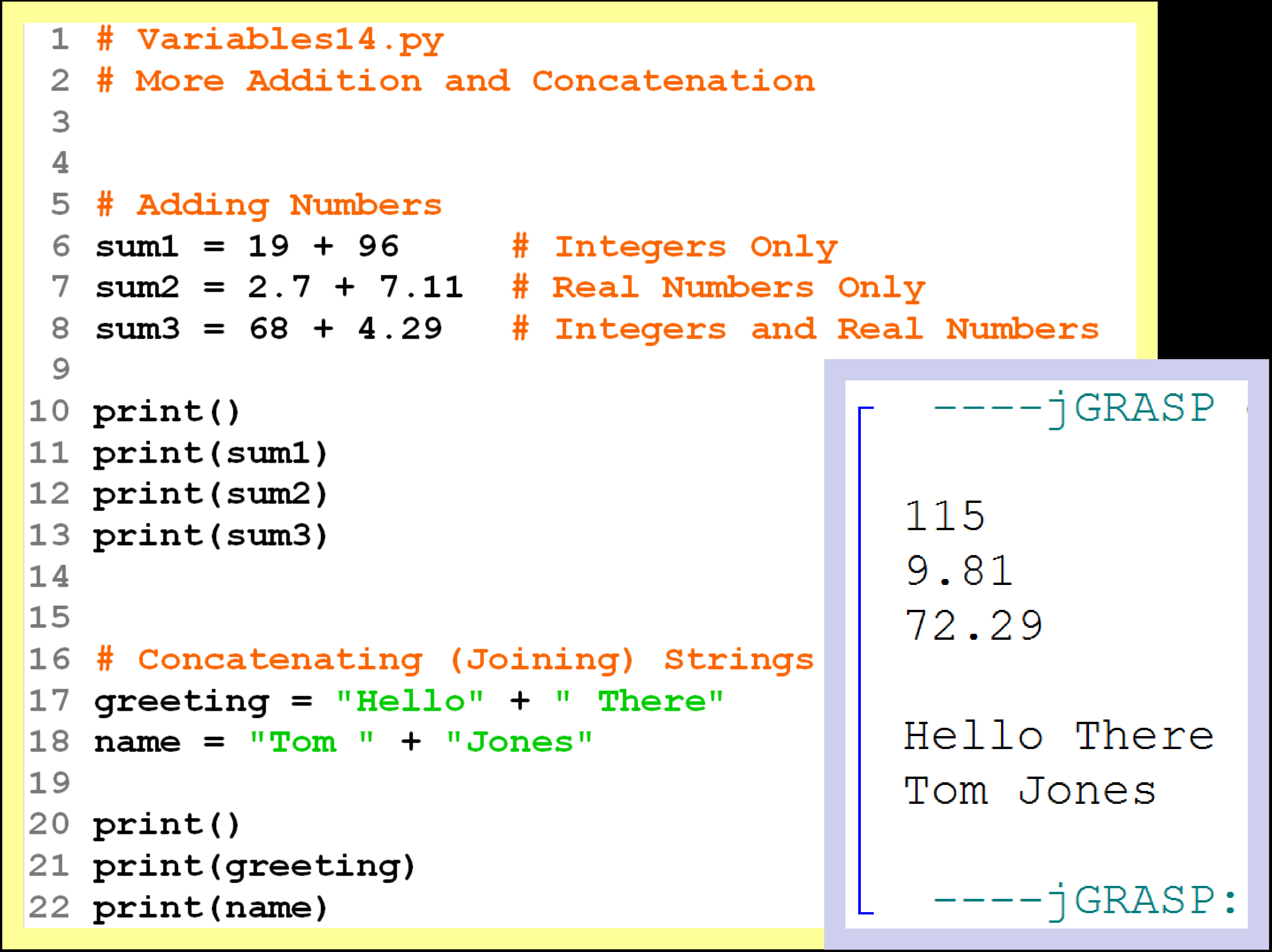
String Concatenation

Concatenation is the joining together of two or more strings.

```
"Hello" + "World"  =  "HelloWorld"

"Hello" + " " + "World"  =  "Hello World"

"100" + "200"  =  "100200"
```

The plus operator (**+**) is used both for *arithmetic addition* and *string concatenation*. The same operator performs two totally different operations. This makes it an *overloaded operator*.

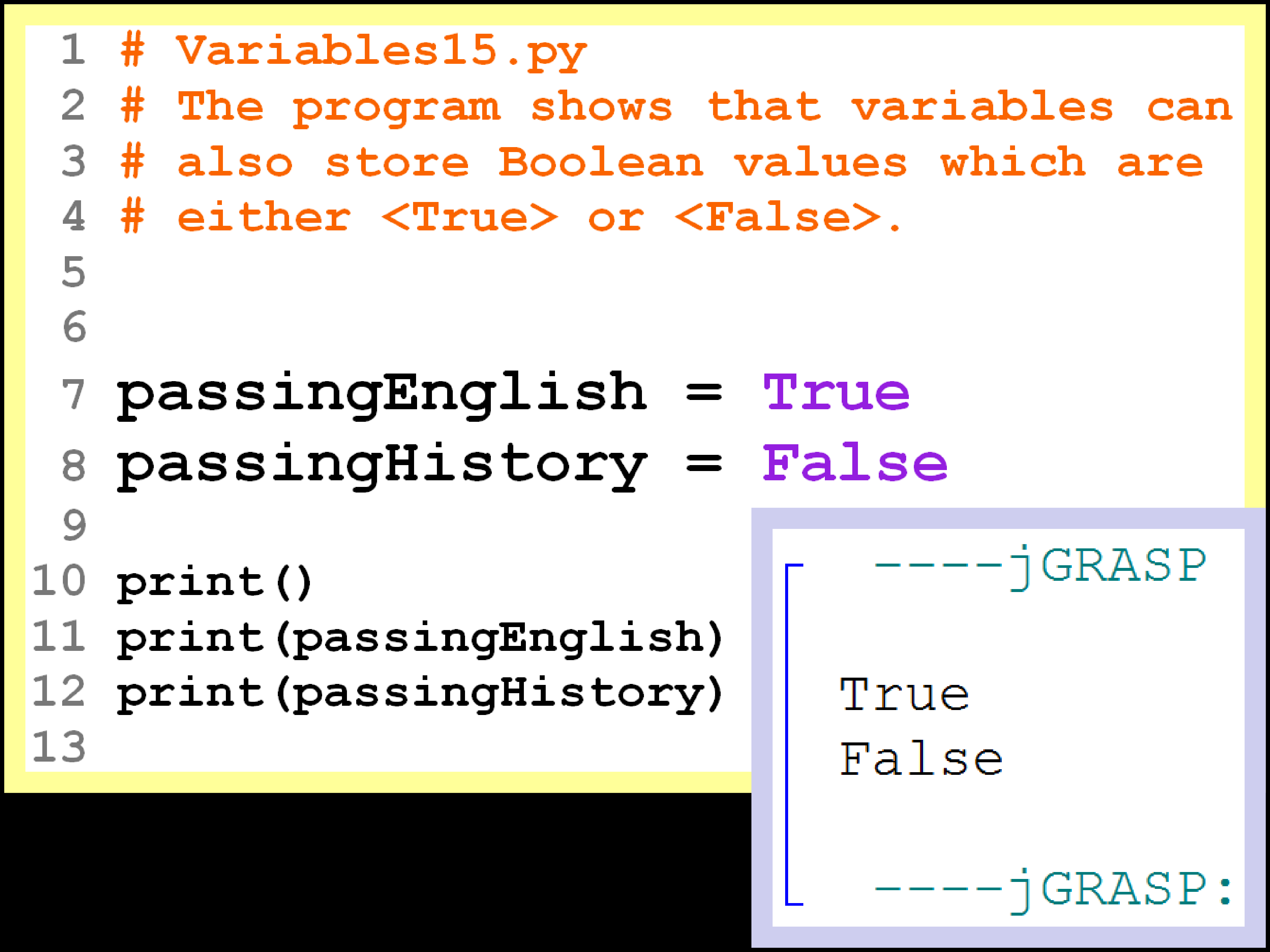
```python
# Variables14.py
# More Addition and Concatenation


# Adding Numbers
sum1 = 19 + 96      # Integers Only
sum2 = 2.7 + 7.11   # Real Numbers Only
sum3 = 68 + 4.29    # Integers and Real Numbers

print()
print(sum1)
print(sum2)
print(sum3)


# Concatenating (Joining) Strings
greeting = "Hello" + " There"
name = "Tom " + "Jones"

print()
print(greeting)
print(name)
```



```python
# Variables14.py
# More Addition and Concatenation


# Adding Numbers
sum1 = 19 + 96      # Integers Only
sum2 = 2.7 + 7.11   # Real Numbers Only
sum3 = 68 + 4.29    # Integers and Real Numbers

print()
print(sum1)
print(sum2)
print(sum3)


# Concatenating (Joining) Strings
greeting = "Hello" + " There"
name = "Tom " + "Jones"

print()
print(greeting)
print(name)
```

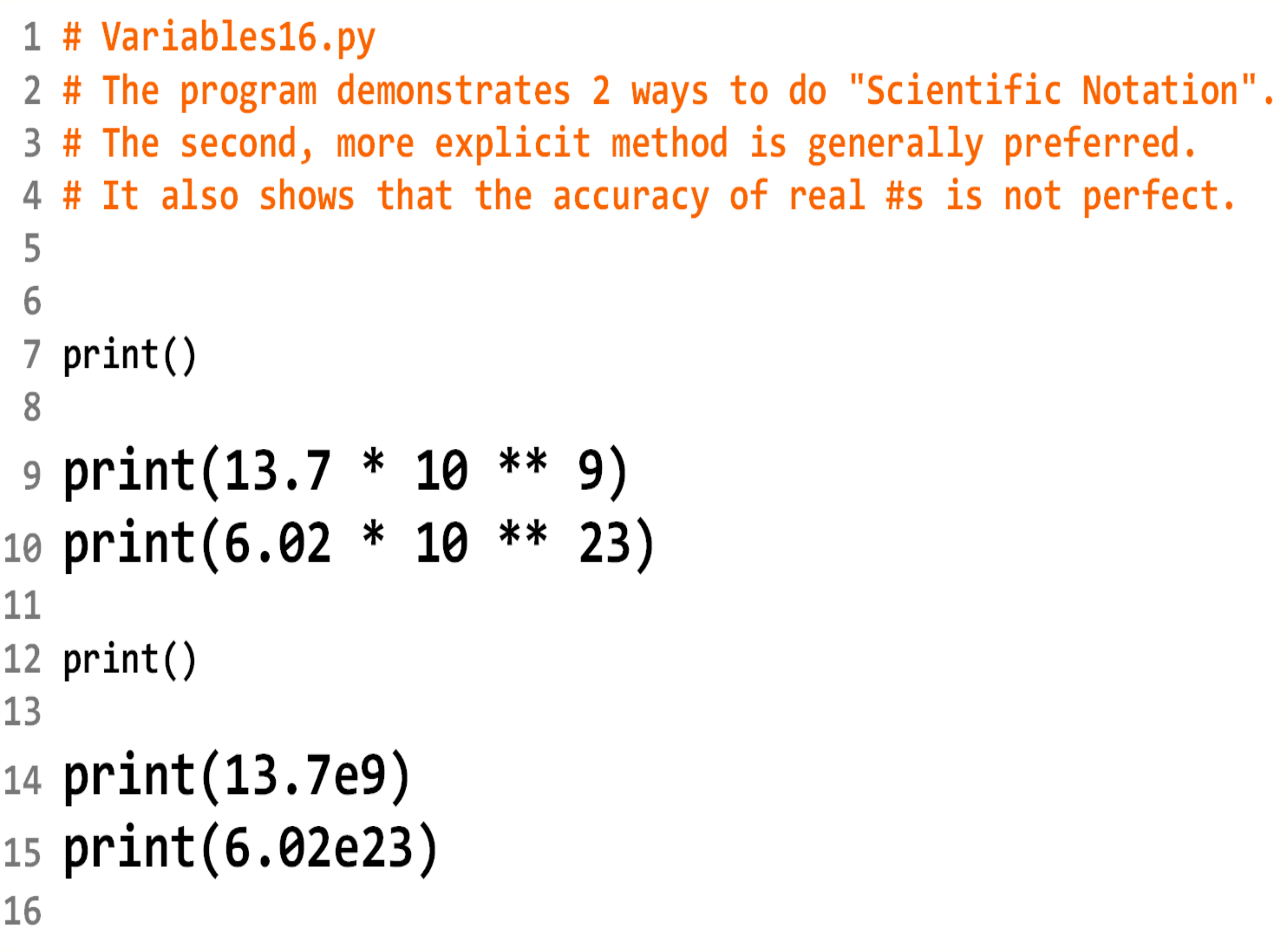
```
 ----jGRASP

115
9.81
72.29

Hello There
Tom Jones

 ----jGRASP:
```

```python
# Variables15.py
# The program shows that variables can
# also store Boolean values which are
# either <True> or <False>.


passingEnglish = True
passingHistory = False

print()
print(passingEnglish)
print(passingHistory)
```



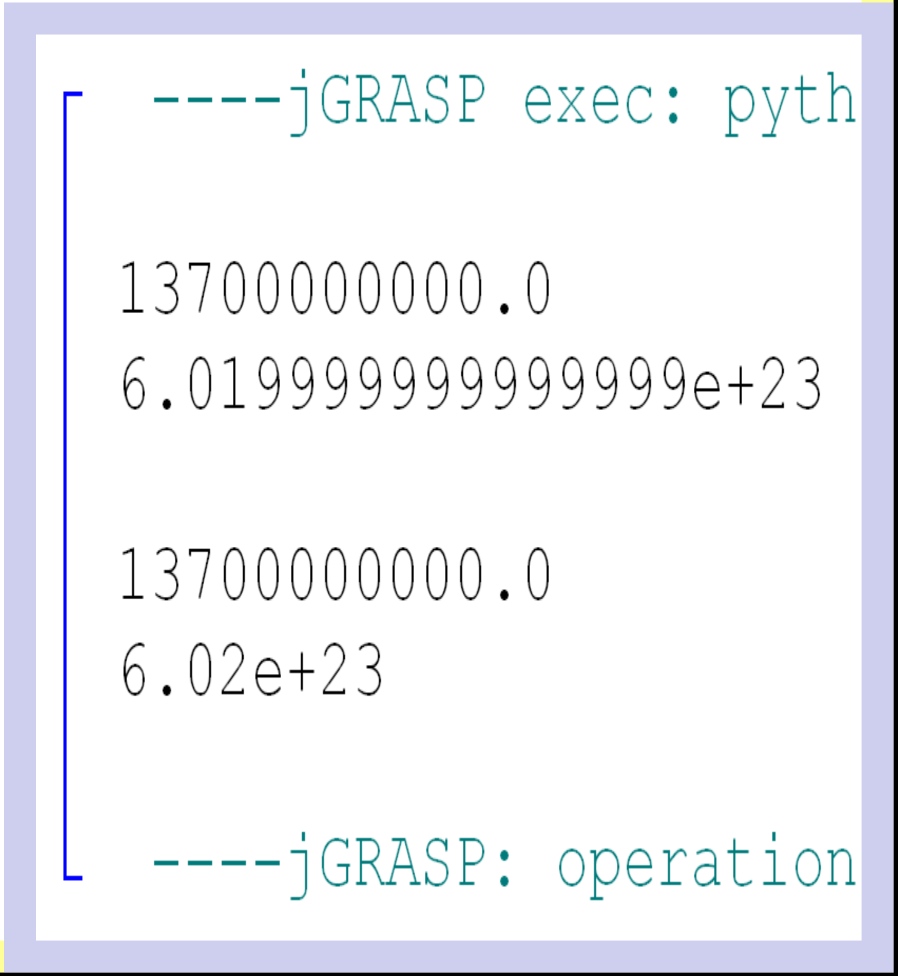
```
# Variables15.py
# The program shows that variables can
# also store Boolean values which are
# either <True> or <False>.


passingEnglish = True
passingHistory = False

print()
print(passingEnglish)
print(passingHistory)

```

```
 ----jGRASP

True
False

 ----jGRASP:
```



```
# Variables16.py
# The program demonstrates 2 ways to do "Scientific Notation".
# The second, more explicit method is generally preferred.
# It also shows that the accuracy of real #s is not perfect.


print()

print(13.7 * 10 ** 9)
print(6.02 * 10 ** 23)

print()

print(13.7e9)
print(6.02e23)

```

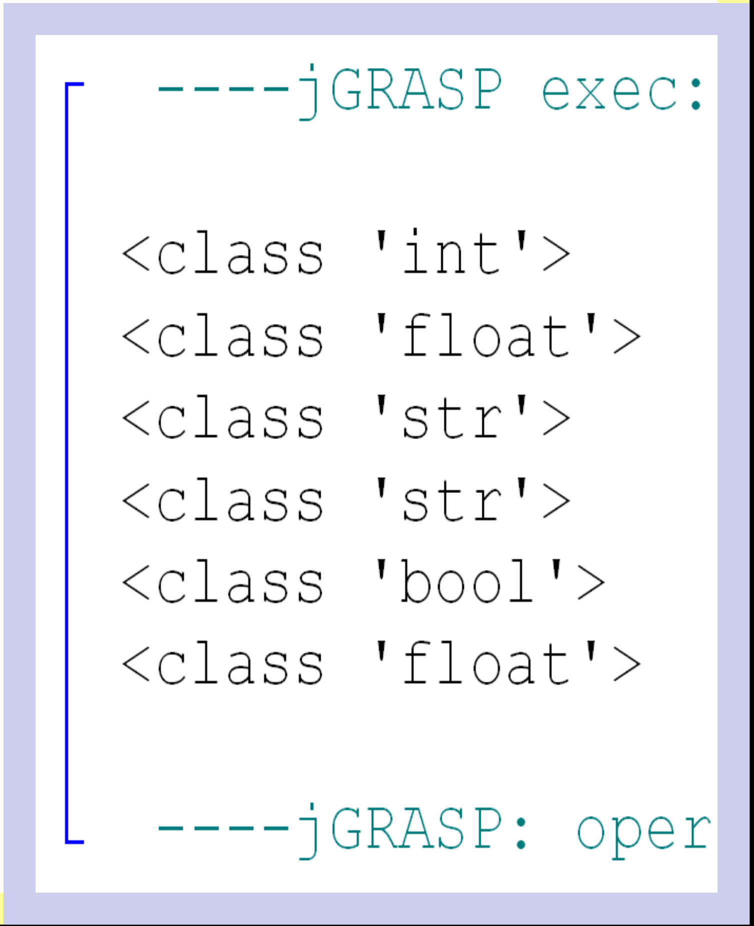
```
# Variables16.py
# The program demonstrates 2 ways to do "Scientific Notation".
# The second, more explicit method is generally preferred.
# It also shows that the accuracy of real #s is not perfect.


print()

print(13.7 * 10 ** 9)
print(6.02 * 10 ** 23)

print()

print(13.7e9)
print(6.02e23)

```



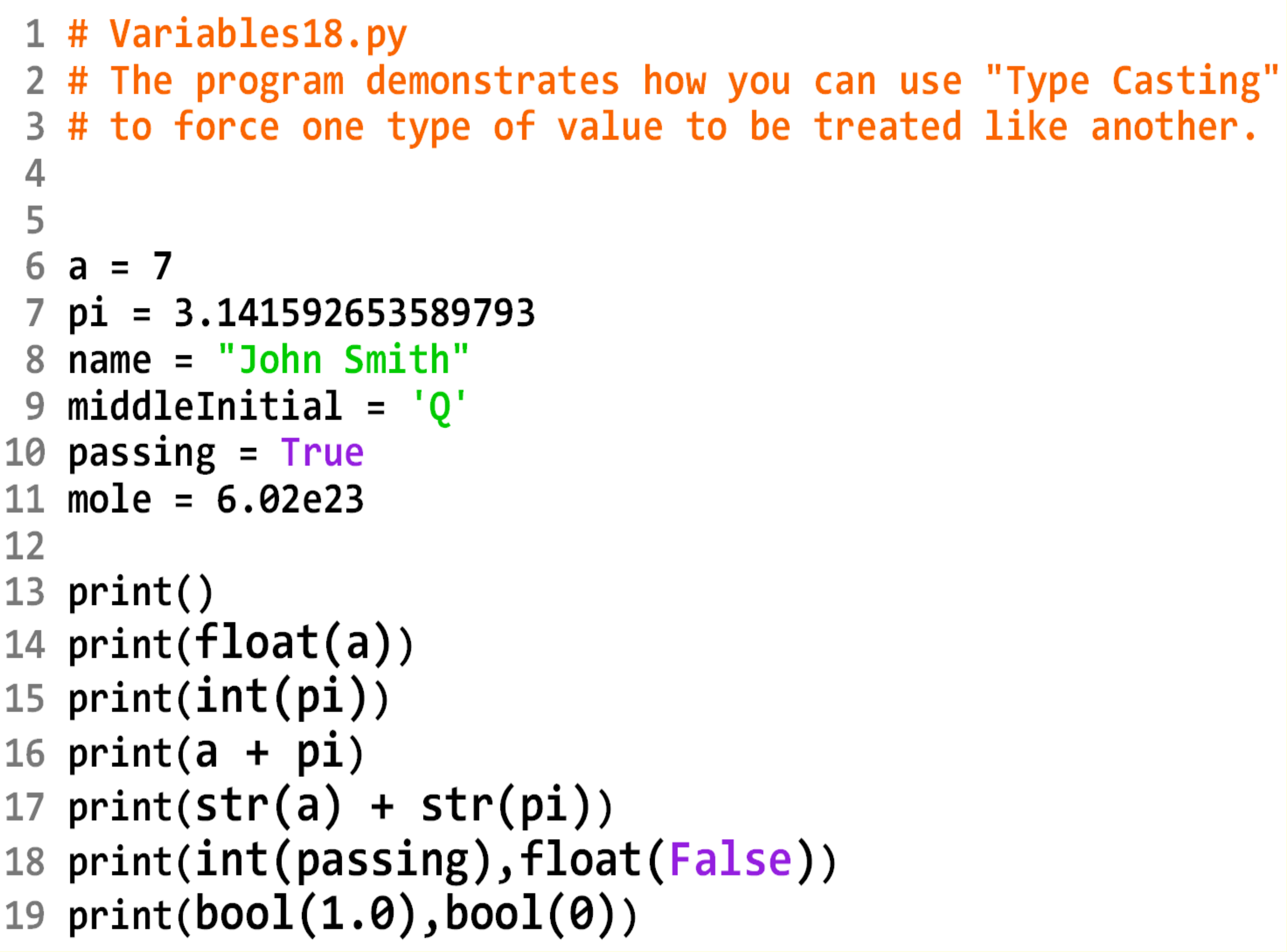
```
 ----jGRASP exec: pyth

13700000000.0
6.019999999999999e+23

13700000000.0
6.02e+23


 ----jGRASP: operation
```

```python
# Variables17.py
# The program demonstrates the <type> command which
# will give you the data type of a particular value.
# NOTE: Python does not distinguish between string
# and character values.  Both are <str>.
# ALSO: Another word for a "real number" is a
# "floating point number".  When scientific notation
# is used, the resulting value is a <float>.


a = 7
pi = 3.141592653589793
name = "John Smith"
middleInitial = 'Q'
passing = True
mole = 6.02e23

print()
print(type(a))
print(type(pi))
print(type(name))
print(type(middleInitial))
print(type(passing))
print(type(mole))
```

```python
# Variables17.py
# The program demonstrates the <type> command which
# will give you the data type of a particular value.
# NOTE: Python does not distinguish between string
# and character values.  Both are <str>.
# ALSO: Another word for a "real number" is a
# "floating point number".  When scientific notation
# is used, the resulting value is a <float>.


a = 7
pi = 3.141592653589793
name = "John Smith"
middleInitial = 'Q'
passing = True
mole = 6.02e23

print()
print(type(a))
print(type(pi))
print(type(name))
print(type(middleInitial))
print(type(passing))
print(type(mole))
```

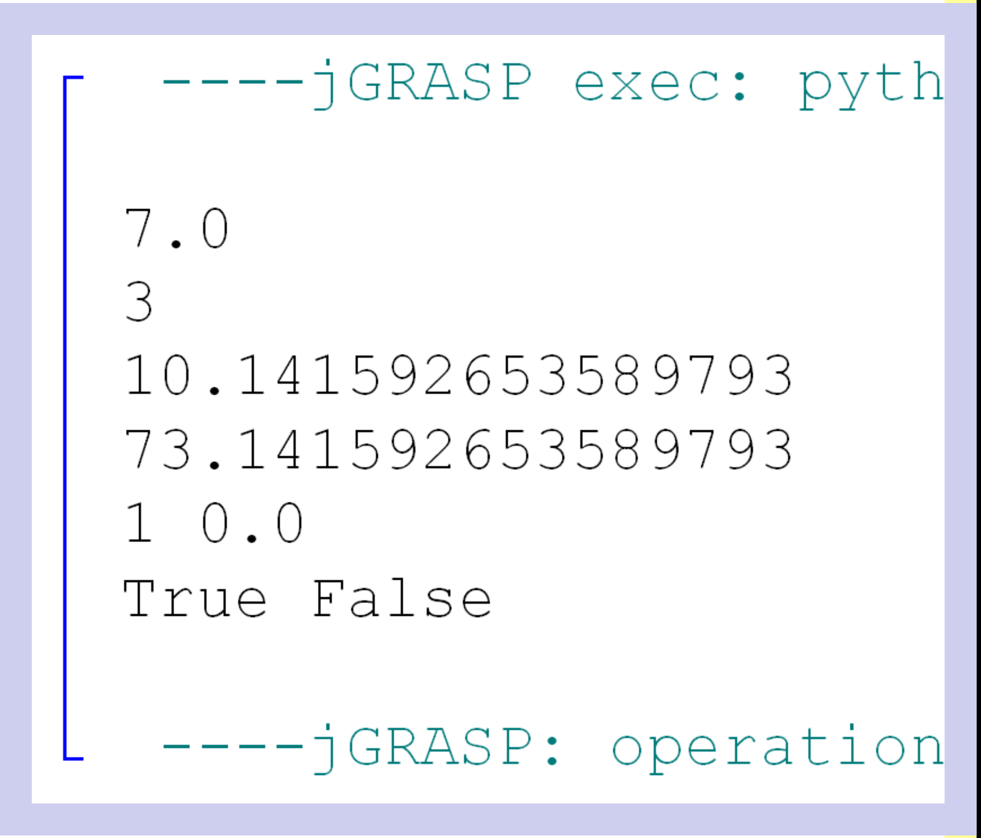


```
 ----jGRASP exec:

<class 'int'>
<class 'float'>
<class 'str'>
<class 'str'>
<class 'bool'>
<class 'float'>

 ----jGRASP: oper
```



```python
# Variables18.py
# The program demonstrates how you can use "Type Casting"
# to force one type of value to be treated like another.


a = 7
pi = 3.14159265358979 3
name = "John Smith"
middleInitial = 'Q'
passing = True
mole = 6.02e23

print()
print(float(a))
print(int(pi))
print(a + pi)
print(str(a) + str(pi))
print(int(passing),float(False))
print(bool(1.0),bool(0))
```

```python
# Variables18.py
# The program demonstrates how you can use "Type Casting"
# to force one type of value to be treated like another.


a = 7
pi = 3.141592653589793
name = "John Smith"
middleInitial = 'Q'
passing = True
mole = 6.02e23

print()
print(float(a))
print(int(pi))
print(a + pi)
print(str(a) + str(pi))
print(int(passing),float(False))
print(bool(1.0),bool(0))
```



```
 ----jGRASP exec: pyth

7.0
3
10.141592653589793
73.141592653589793
1 0.0
True False

 ----jGRASP: operation
```

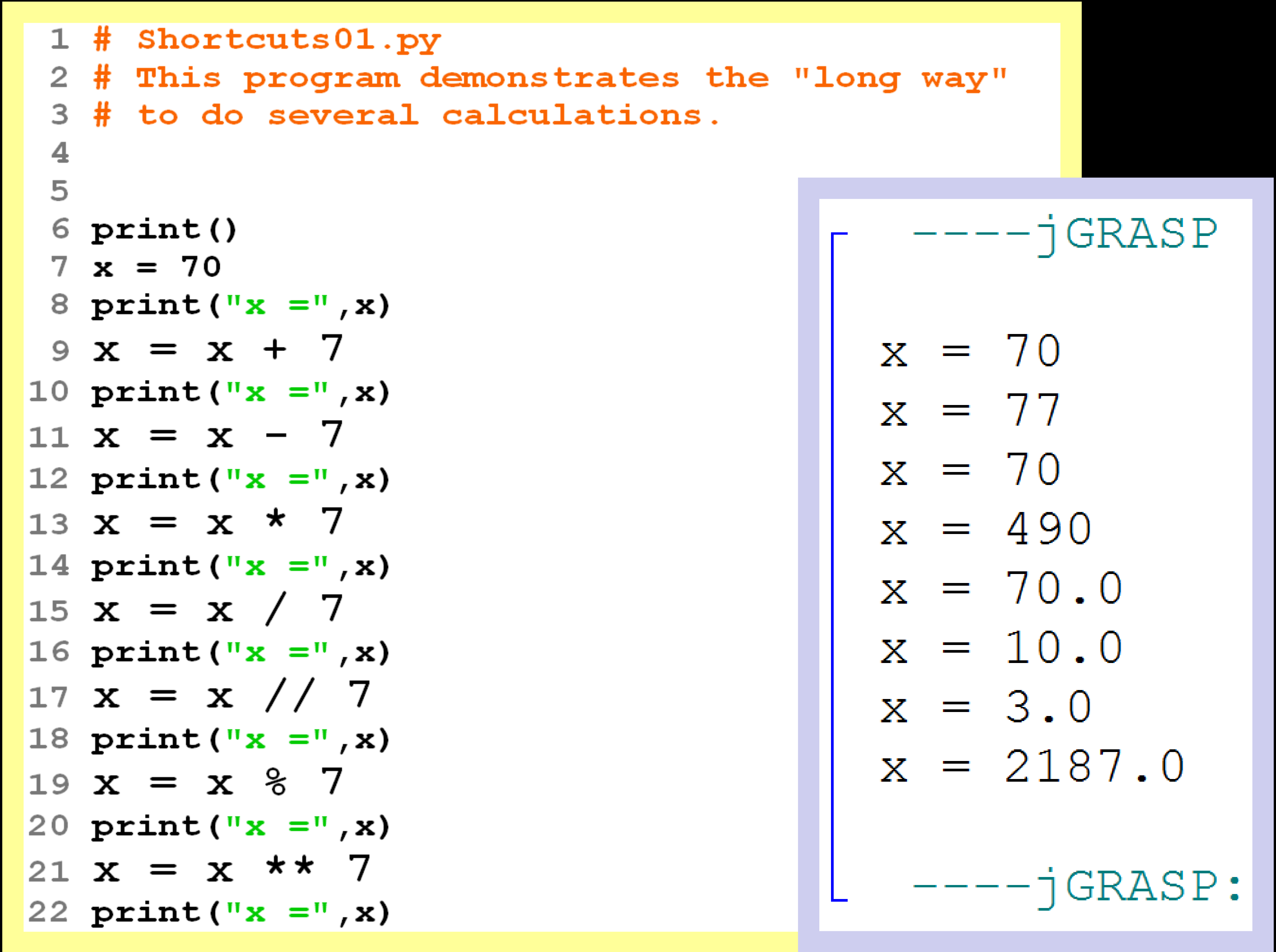
Don't Get Confused!

Value	Data Type
7	int
7.0	float
"7"	str
'7'	str

Section 4.5

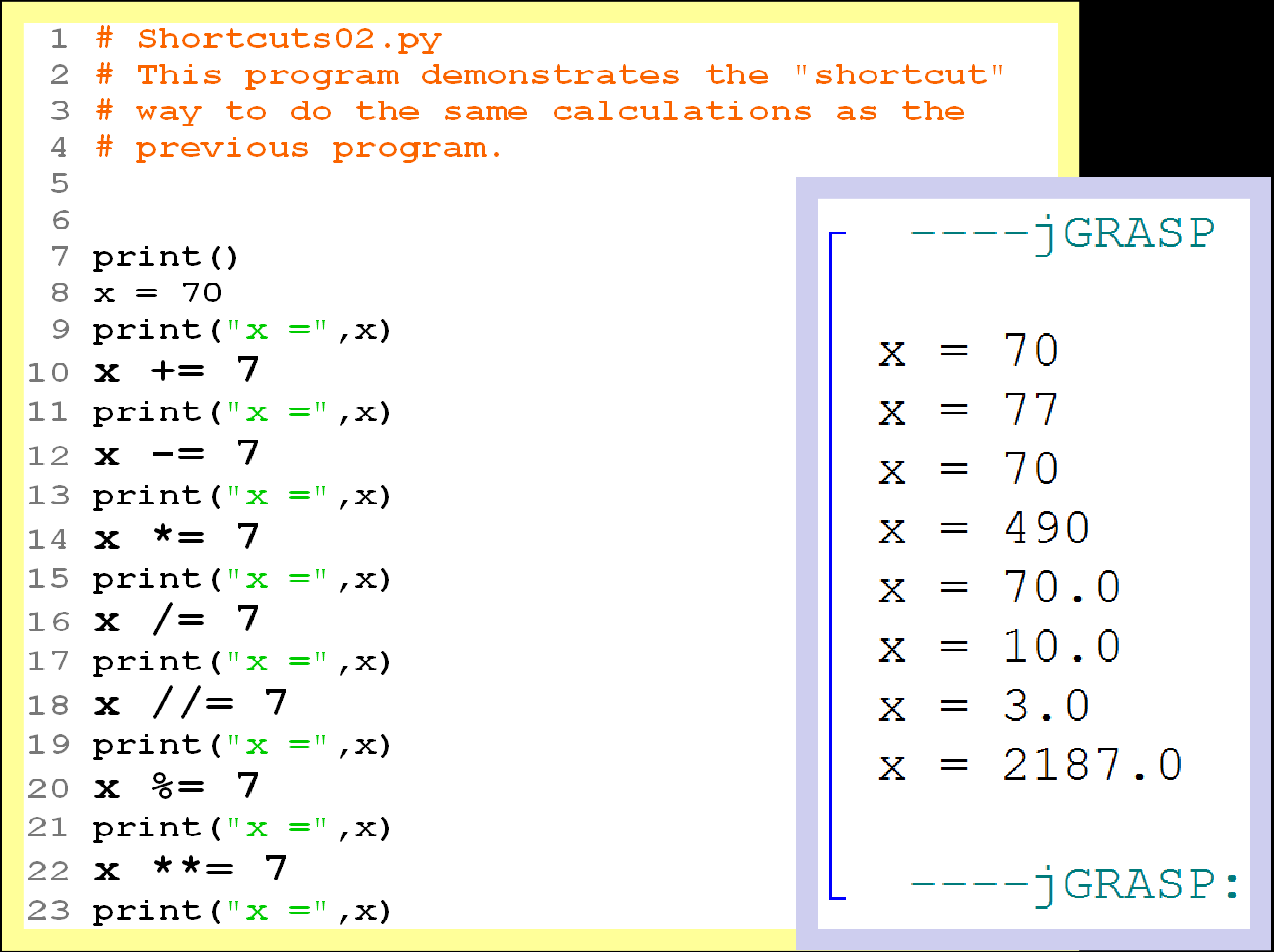
Shortcuts

```python
# Shortcuts01.py
# This program demonstrates the "long way"
# to do several calculations.


print()
x = 70
print("x =",x)
x = x + 7
print("x =",x)
x = x - 7
print("x =",x)
x = x * 7
print("x =",x)
x = x / 7
print("x =",x)
x = x // 7
print("x =",x)
x = x % 7
print("x =",x)
x = x ** 7
print("x =",x)
```



```python
# Shortcuts01.py
# This program demonstrates the "long way"
# to do several calculations.


print()
x = 70
print("x =",x)
x = x + 7
print("x =",x)
x = x - 7
print("x =",x)
x = x * 7
print("x =",x)
x = x / 7
print("x =",x)
x = x // 7
print("x =",x)
x = x % 7
print("x =",x)
x = x ** 7
print("x =",x)
```

```
 ----jGRASP

x = 70
x = 77
x = 70
x = 490
x = 70.0
x = 10.0
x = 3.0
x = 2187.0

 ----jGRASP:
```



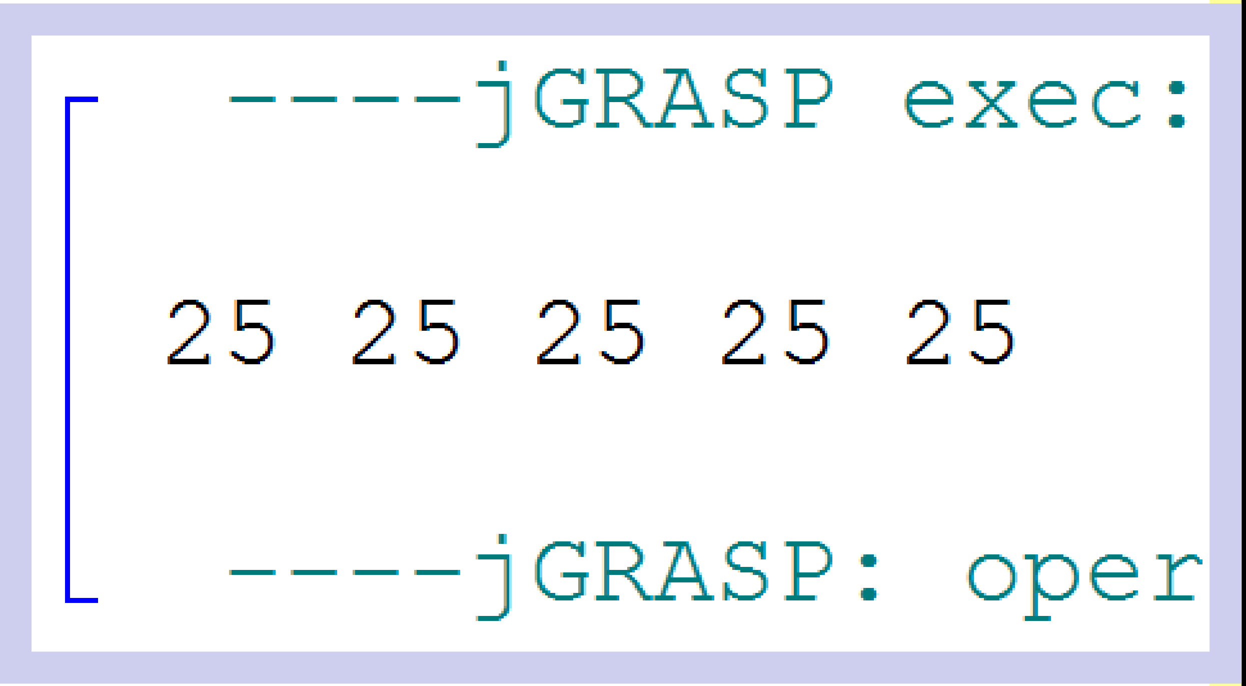
```python
# Shortcuts02.py
# This program demonstrates the "shortcut"
# way to do the same calculations as the
# previous program.


print()
x = 70
print("x =",x)
x += 7
print("x =",x)
x -= 7
print("x =",x)
x *= 7
print("x =",x)
x /= 7
print("x =",x)
x //= 7
print("x =",x)
x %= 7
print("x =",x)
x **= 7
print("x =",x)
```

```
 ----jGRASP

x = 70
x = 77
x = 70
x = 490
x = 70.0
x = 10.0
x = 3.0
x = 2187.0

 ----jGRASP:
```

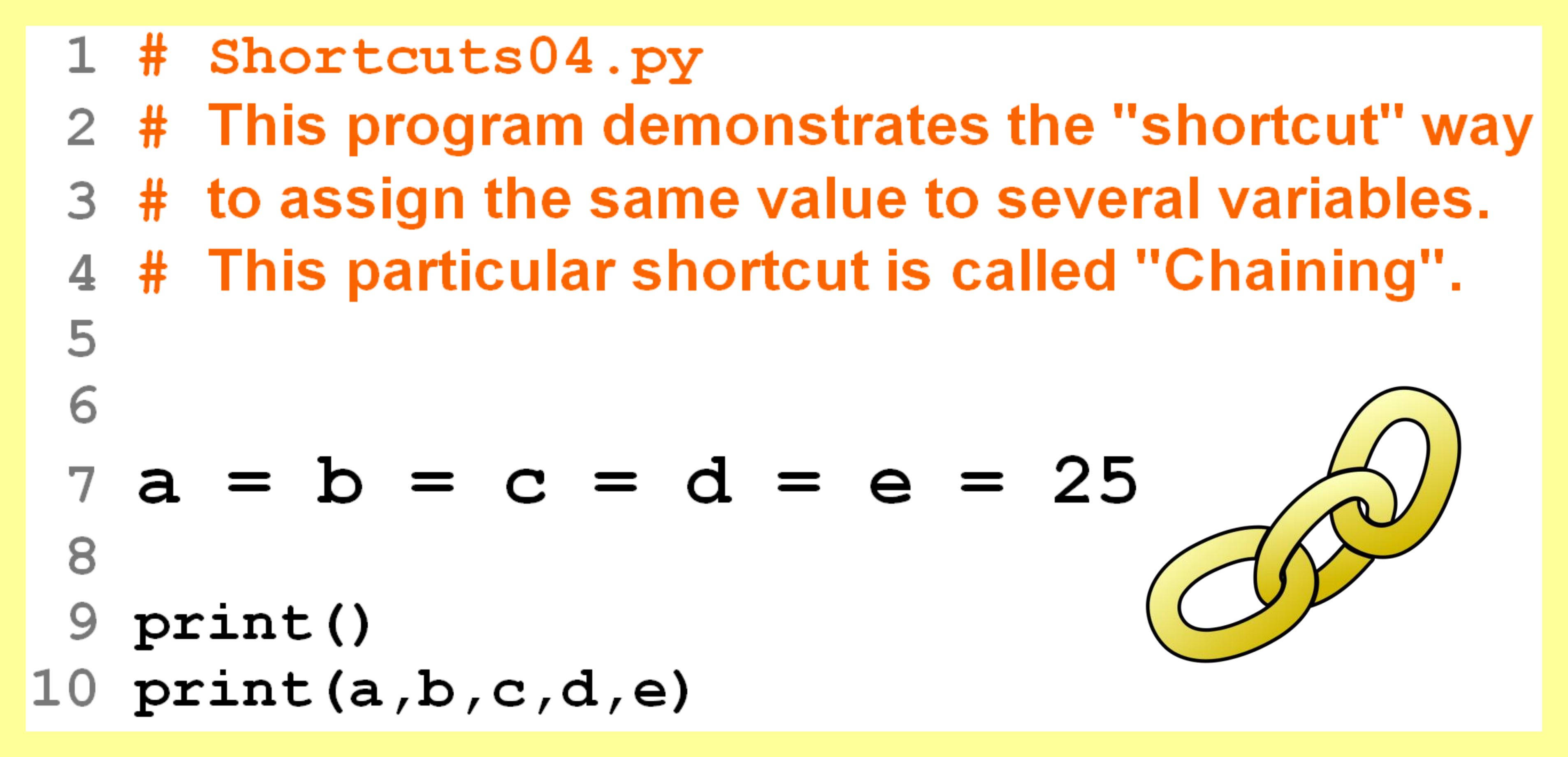
Arithmetic Operator Shortcuts

Long Way	Shortcut	Meaning
x = x + 7	x += 7	Add **7** to the current value of **x**.
x = x – 7	x -= 7	Subtract **7** from the current value of **x**.
x = x * 7	x *= 7	Multiply the current value of **x** by **7**.
x = x / 7	x /= 7	Divide the current value of **x** by **7** using real number division.
x = x // 7	x //= 7	Divide the current value of **x** by **7** using integer division.
x = x % 7	x %= 7	Divide the current value of **x** by **7** using remainder division.
x = x ** 7	x **= 7	Take the current value of **x** to the **7**th power.

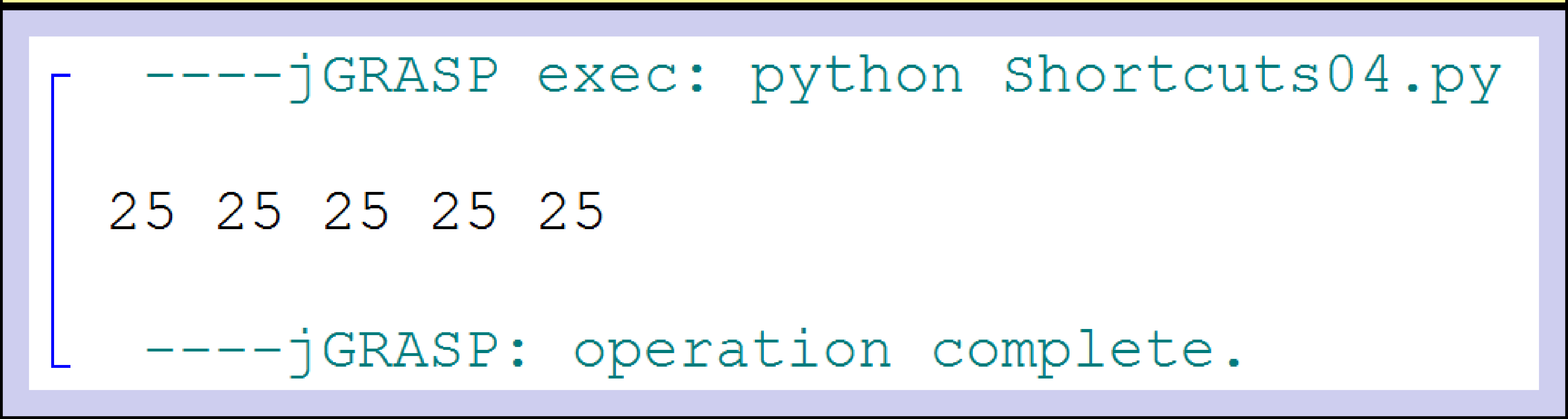
```python
# Shortcuts03.py
# This program demonstrates the "long way" to
# assign the same value to several variables.


a = 25
b = 25
c = 25
d = 25
e = 25

print()
print(a,b,c,d,e)

```



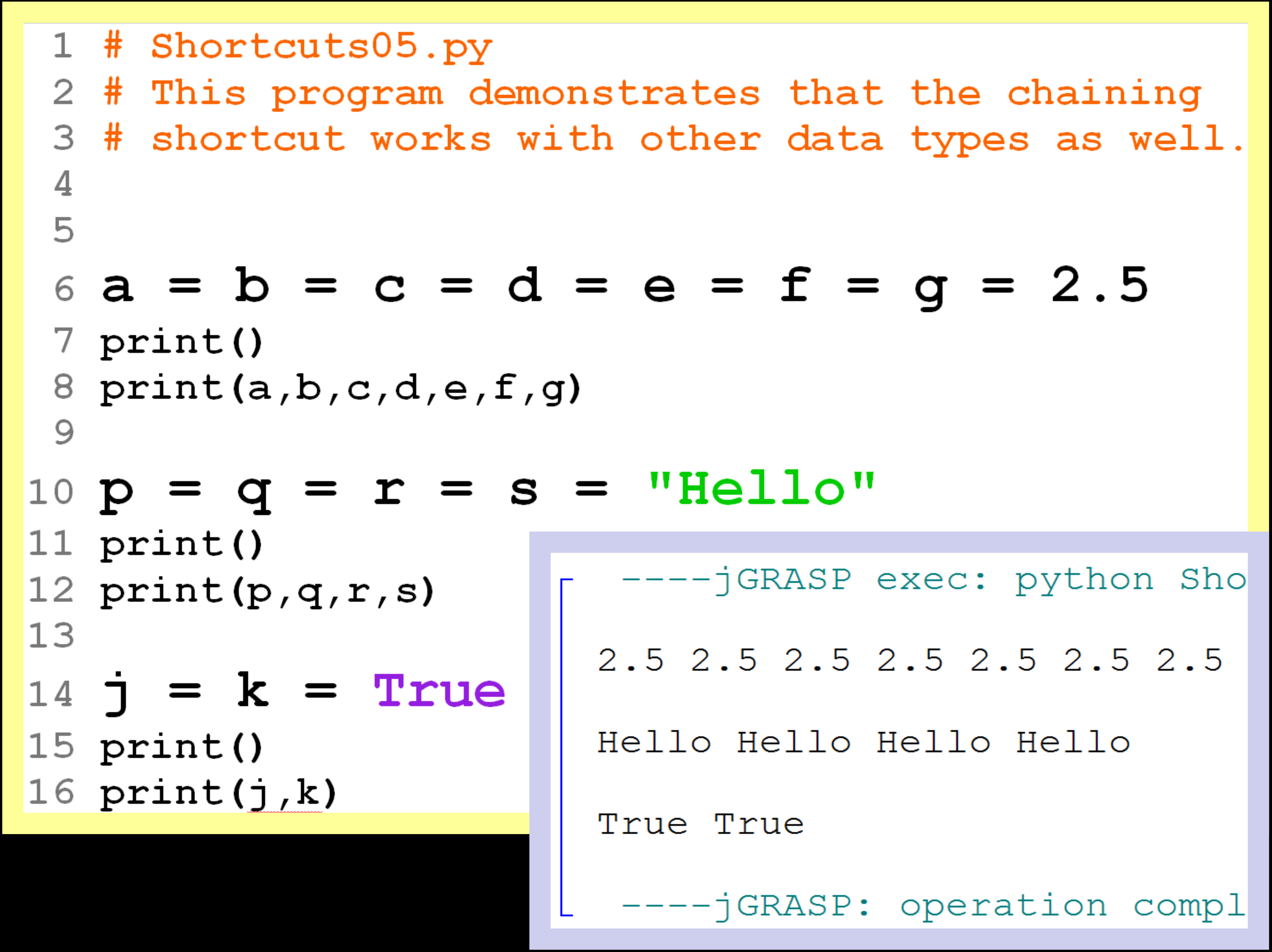
```
 ----jGRASP exec:

25 25 25 25 25

 ----jGRASP: oper
```



```python
#  Shortcuts04.py
#  This program demonstrates the "shortcut" way
#  to assign the same value to several variables.
#  This particular shortcut is called "Chaining".


a = b = c = d = e = 25

print()
print(a,b,c,d,e)
```



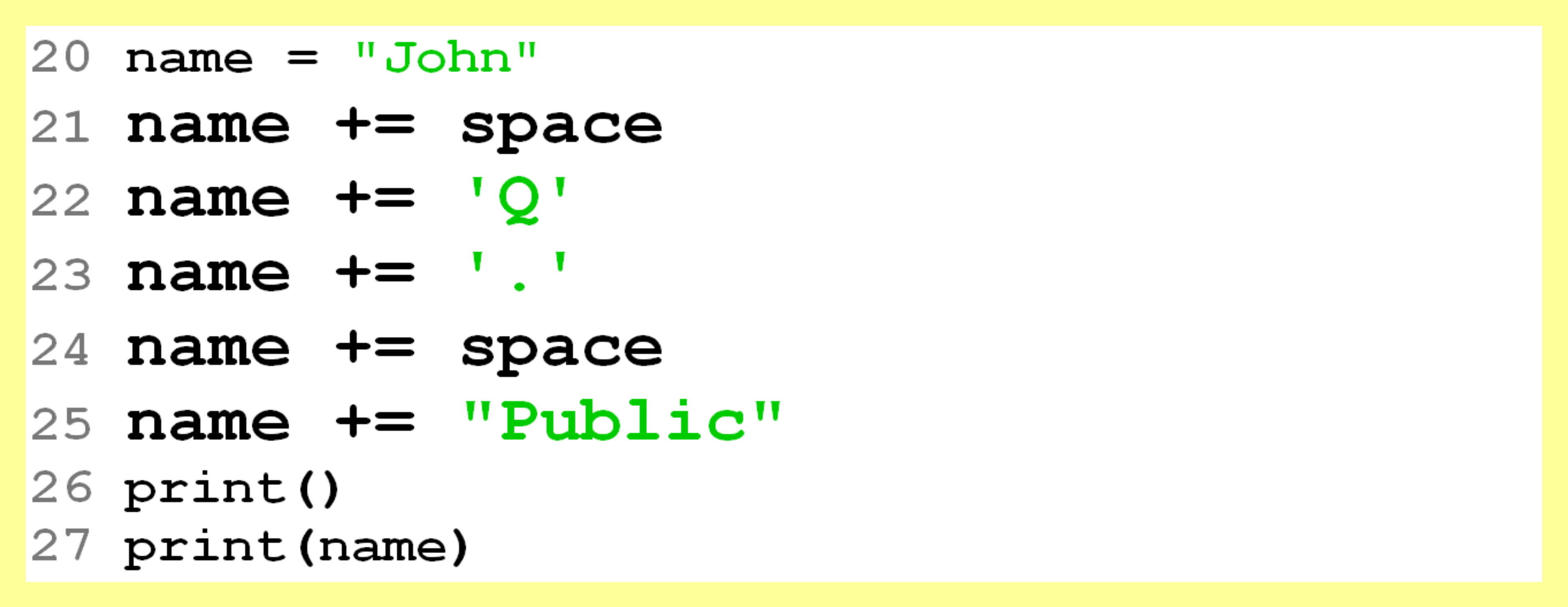
```
 ----jGRASP exec: python Shortcuts04.py

25 25 25 25 25

 ----jGRASP: operation complete.
```



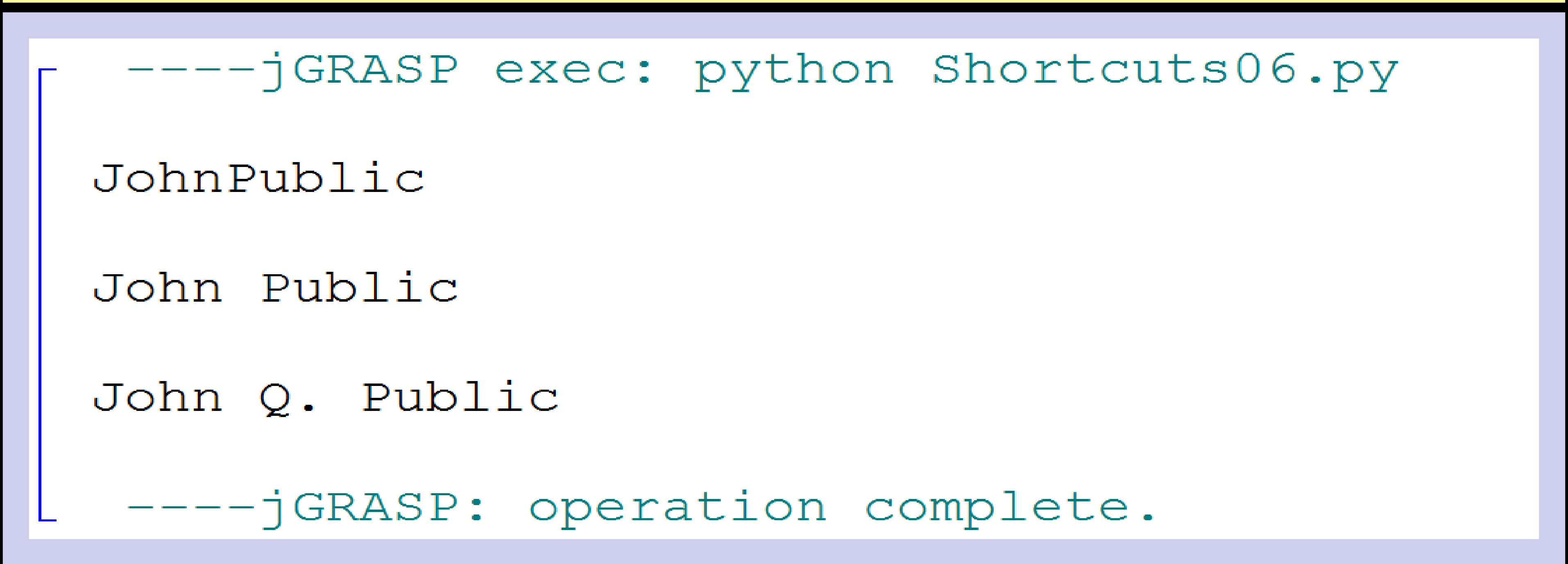
```
# Shortcuts05.py
# This program demonstrates that the chaining
# shortcut works with other data types as well.


a = b = c = d = e = f = g = 2.5
print()
print(a,b,c,d,e,f,g)

p = q = r = s = "Hello"
print()
print(p,q,r,s)

j = k = True
print()
print(j,k)
```

```
 ----jGRASP exec: python Sho

2.5 2.5 2.5 2.5 2.5 2.5 2.5

Hello Hello Hello Hello

True True

 ----jGRASP: operation compl
```

```python
# Shortcuts06.py
# This program demonstrates that the <+=> shortcut
# can be used with strings to join a string value
# to the end of an existing string.
# This means that <+=> is also an "Overloaded Operator".


name = "John"
name += "Public"
print()
print(name)

name = "John"
space = ' '
name += space
name += "Public"
print()
print(name)
```



```python
name = "John"
name += space
name += 'Q'
name += '.'
name += space
name += "Public"
print()
print(name)
```

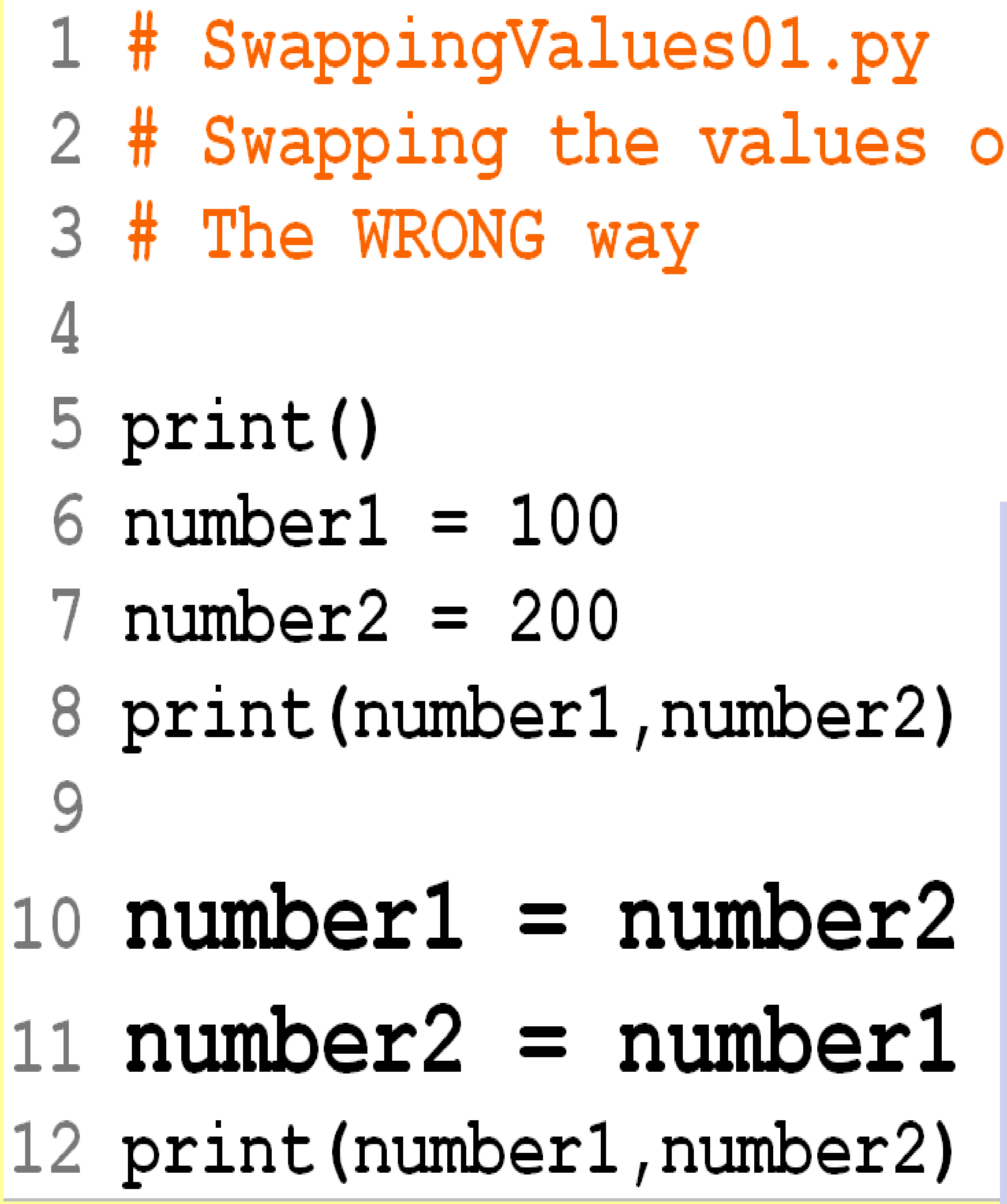


```
 ----jGRASP exec: python Shortcuts06.py

JohnPublic

John Public

John Q. Public

 ----jGRASP: operation complete.
```

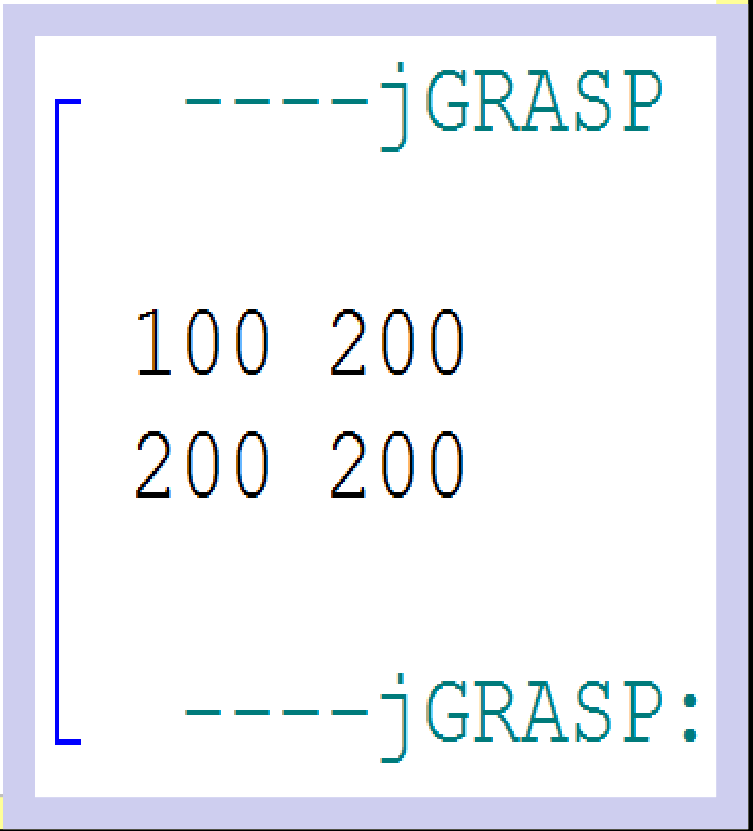
Section 4.6

Swapping Variable Values

```
# SwappingValues01.py
# Swapping the values of 2 variables
# The WRONG way

print()
number1 = 100
number2 = 200
print(number1,number2)

number1 = number2
number2 = number1
print(number1,number2)
```



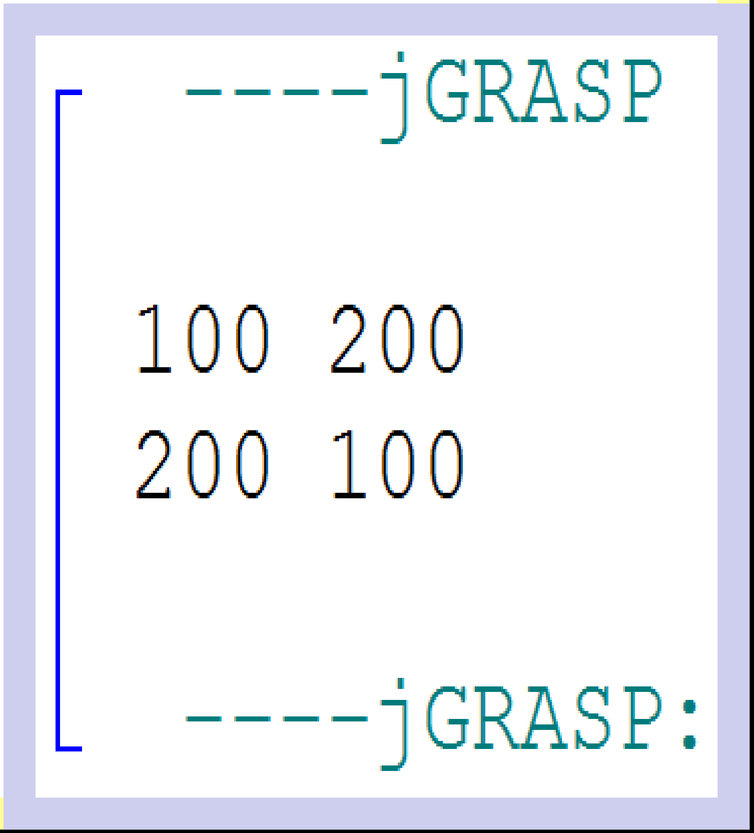
```python
# SwappingValues01.py
# Swapping the values of 2 variables
# The WRONG way

print()
number1 = 100
number2 = 200
print(number1,number2)

number1 = number2
number2 = number1
print(number1,number2)
```



```
  ----jGRASP

100 200
200 200

  ----jGRASP:
```

```python
# SwappingValues02.py
# Swapping the values of 2 variables
# Using a temporary variable
# This technique works in any language.


print()
number1 = 100
number2 = 200
print(number1,number2)

temp = number1
number1 = number2
number2 = temp
print(number1,number2)
```

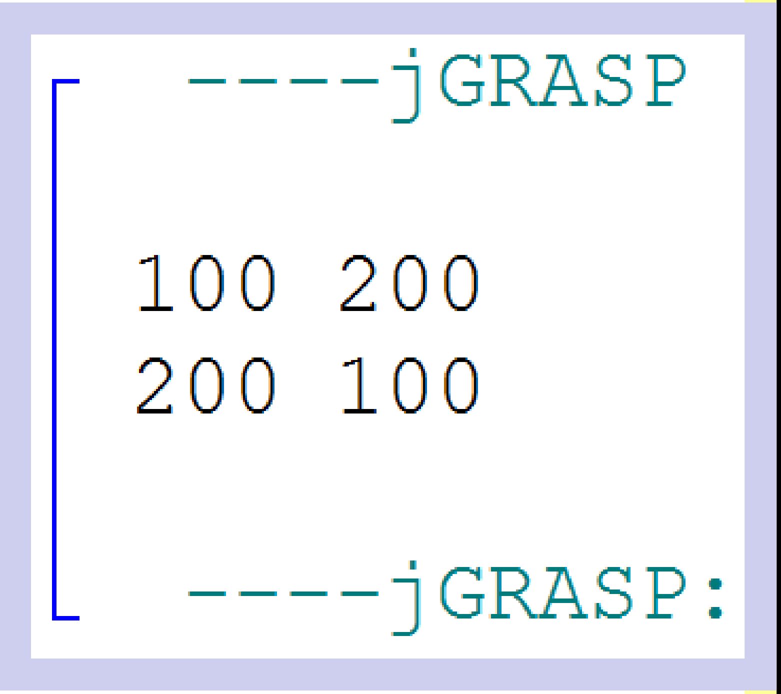
```python
# SwappingValues02.py
# Swapping the values of 2 variables
# Using a temporary variable
# This technique works in any language.


print()
number1 = 100
number2 = 200
print(number1,number2)

temp = number1
number1 = number2
number2 = temp
print(number1,number2)
```



```
 ----jGRASP


100 200
200 100


 ----jGRASP:
```

```python
# SwappingValues03.py
# Swapping the values of 2 variables
# Using "Simultaneous Assignment"
# This "shortcut" only works in Python.


print()
number1 = 100
number2 = 200
print(number1,number2)

number1,number2 = number2,number1
print(number1,number2)
```

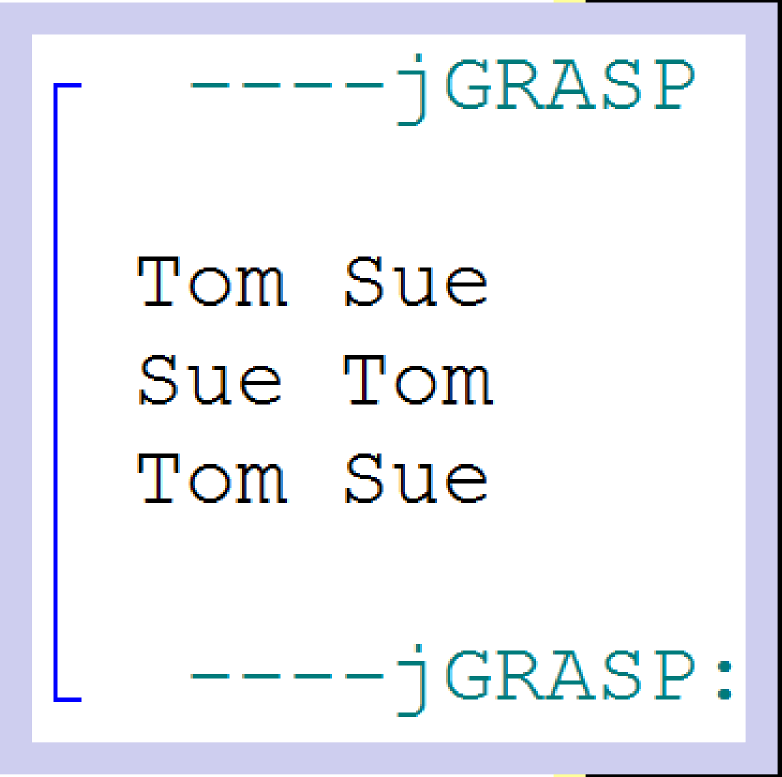
```python
# SwappingValues03.py
# Swapping the values of 2 variables
# Using "Simultaneous Assignment"
# This "shortcut" only works in Python.


print()
number1 = 100
number2 = 200
print(number1,number2)

number1,number2 = number2,number1
print(number1,number2)
```



```
 ----jGRASP


 100 200
 200 100


 ----jGRASP:
```

```python
# SwappingValues04.py
# Swapping the values of 2 string variables
# You can swap other data types as well.
# NOTE: If you swap 2 variables twice, they
# wind up with their original values.


print()
name1 = "Tom"
name2 = "Sue"
print(name1,name2)

# first swap
temp = name1
name1 = name2
name2 = temp
print(name1,name2)

# second swap
name1,name2 = name2,name1
print(name1,name2)
```

```python
# SwappingValues04.py
# Swapping the values of 2 string variables
# You can swap other data types as well.
# NOTE: If you swap 2 variables twice, they
# wind up with their original values.


print()
name1 = "Tom"
name2 = "Sue"
print(name1,name2)

# first swap
temp = name1
name1 = name2
name2 = temp
print(name1,name2)

# second swap
name1,name2 = name2,name1
print(name1,name2)
```

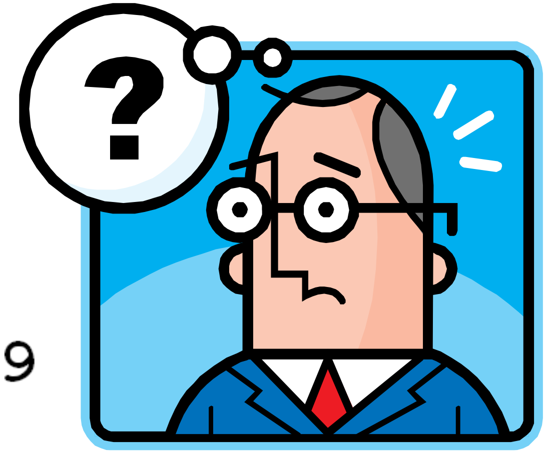


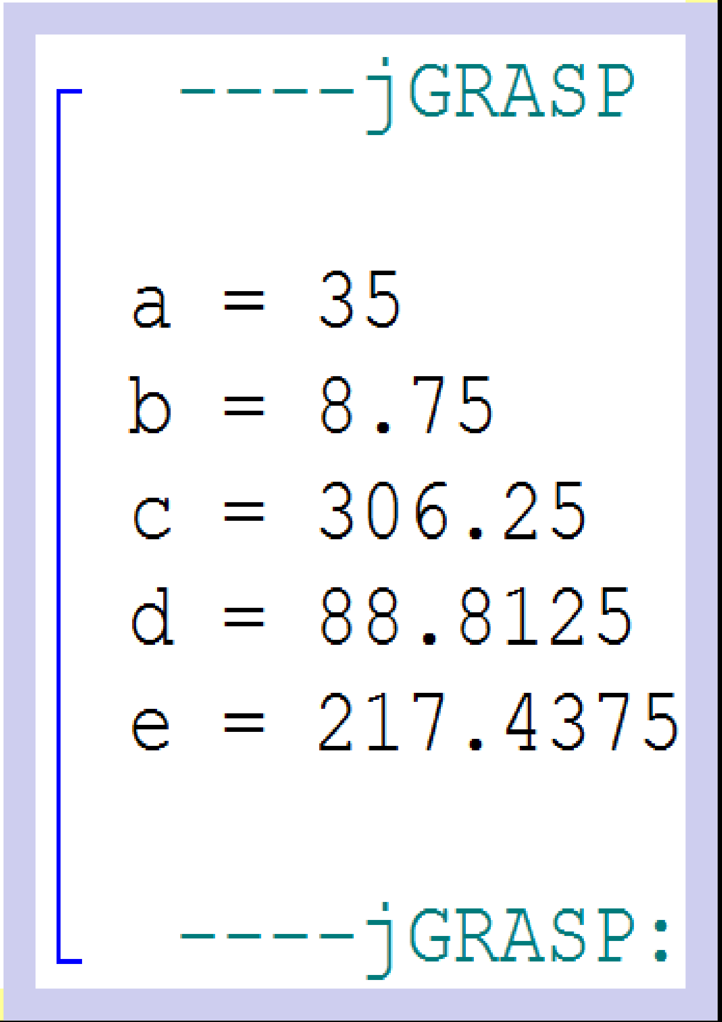
```
 ----jGRASP


Tom Sue
Sue Tom
Tom Sue


 ----jGRASP:
```

Section 4.7

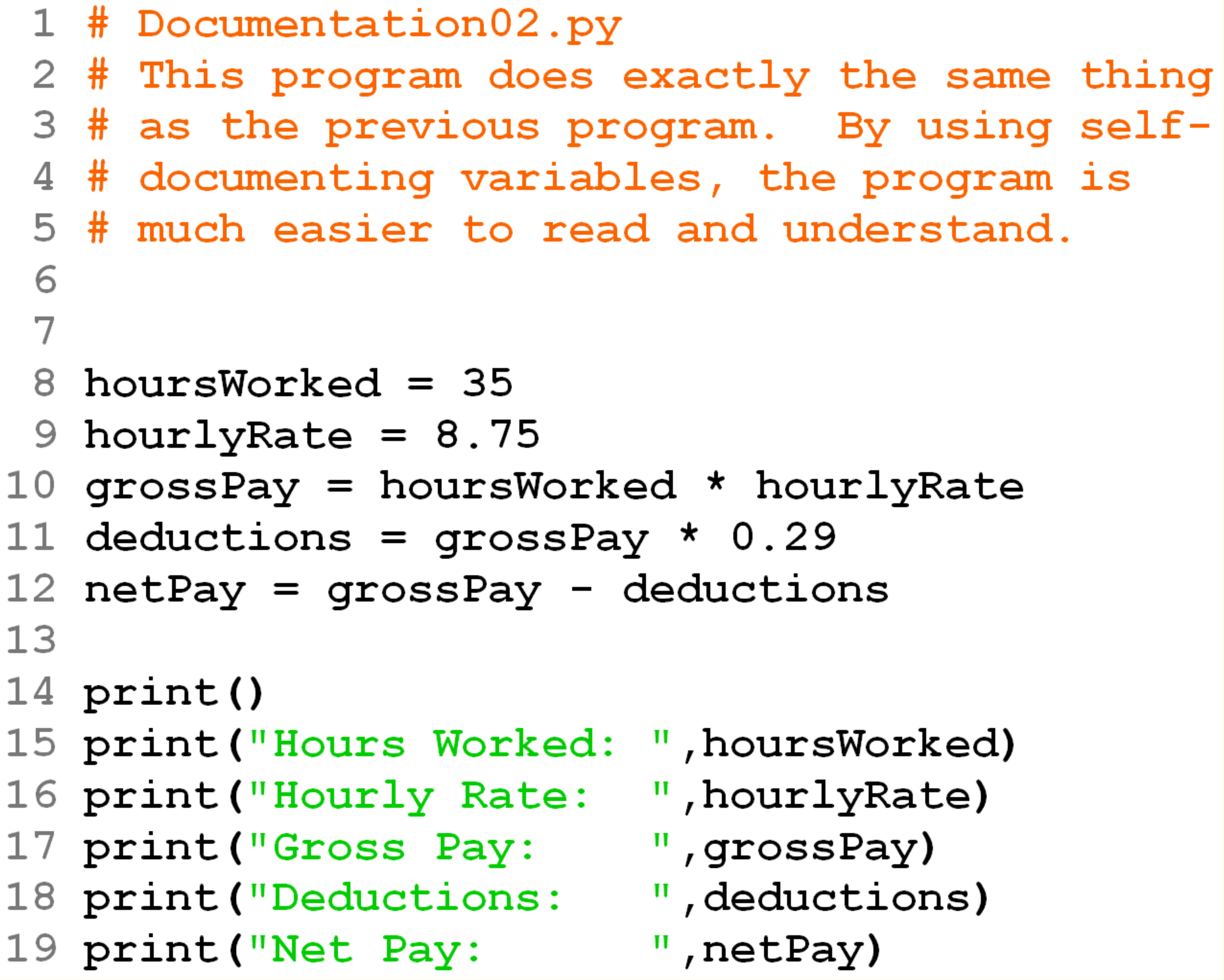
Documenting Your Programs

```python
# Documentation01.py
# This is an example of a poorly written
# program with single-letter variables.
# Do you have any idea what this program does?


a = 35
b = 8.75
c = a * b
d = c * 0.29
e = c - d

print()
print("a =",a)
print("b =",b)
print("c =",c)
print("d =",d)
print("e =",e)
```

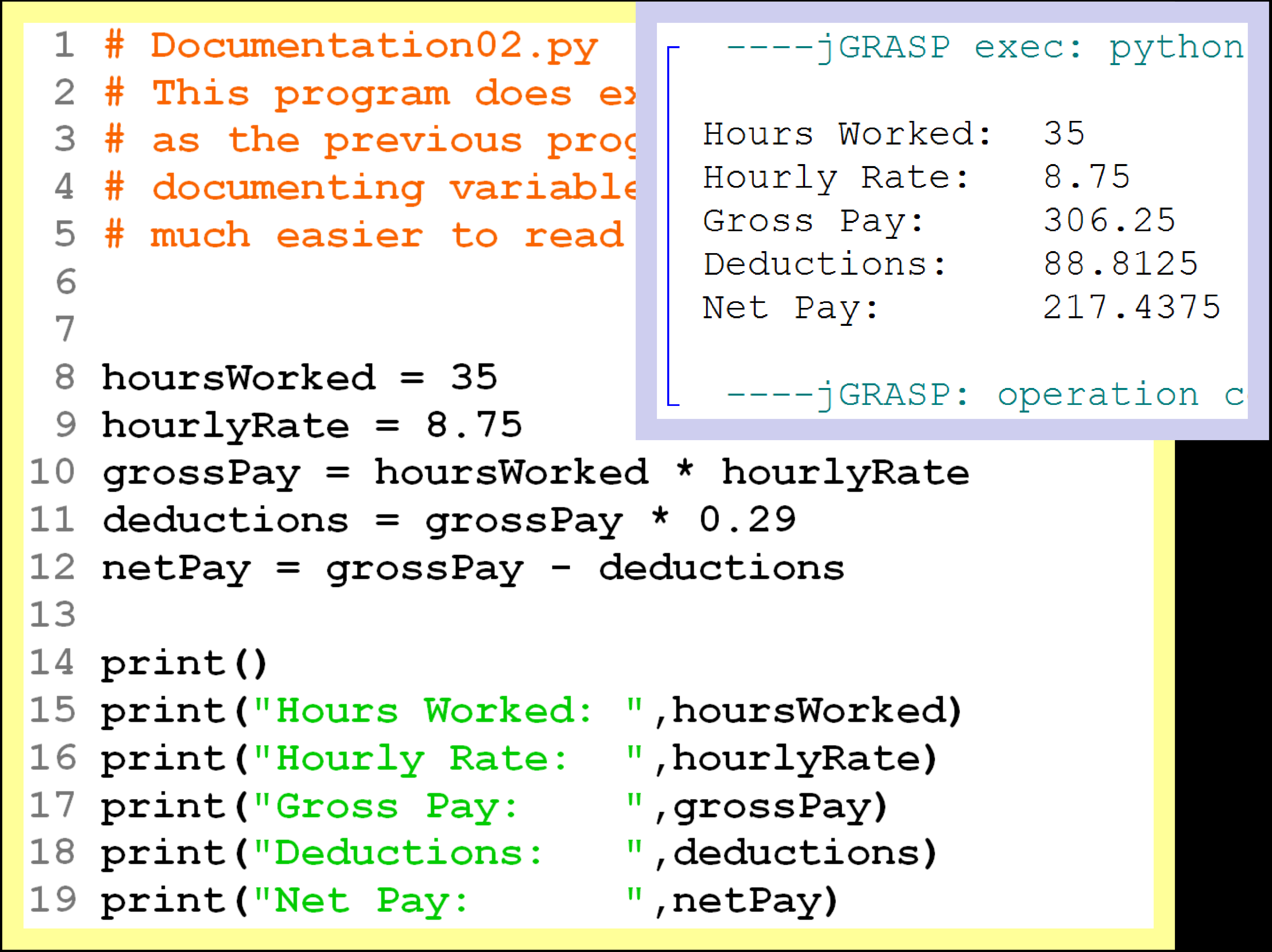




```
 ----jGRASP

 a = 35
 b = 8.75
 c = 306.25
 d = 88.8125
 e = 217.4375

 ----jGRASP:
```

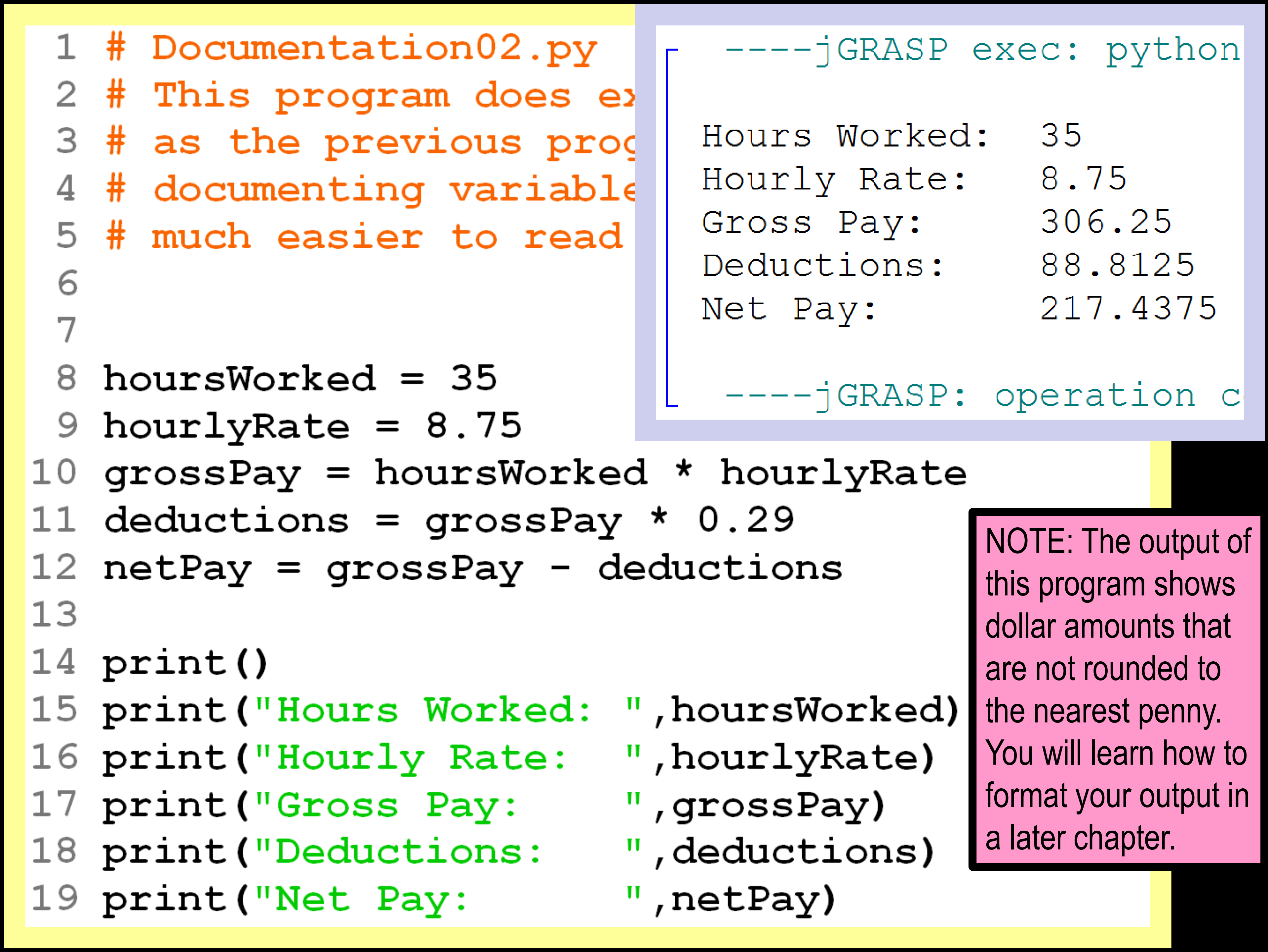


```python
# Documentation02.py
# This program does exactly the same thing
# as the previous program.  By using self-
# documenting variables, the program is
# much easier to read and understand.


hoursWorked = 35
hourlyRate = 8.75
grossPay = hoursWorked * hourlyRate
deductions = grossPay * 0.29
netPay = grossPay - deductions

print()
print("Hours Worked: ",hoursWorked)
print("Hourly Rate:  ",hourlyRate)
print("Gross Pay:    ",grossPay)
print("Deductions:   ",deductions)
print("Net Pay:      ",netPay)
```



```python
# Documentation02.py
# This program does ex
# as the previous prog
# documenting variable
# much easier to read


hoursWorked = 35
hourlyRate = 8.75
grossPay = hoursWorked * hourlyRate
deductions = grossPay * 0.29
netPay = grossPay - deductions

print()
print("Hours Worked:  ",hoursWorked)
print("Hourly Rate:   ",hourlyRate)
print("Gross Pay:     ",grossPay)
print("Deductions:    ",deductions)
print("Net Pay:       ",netPay)
```

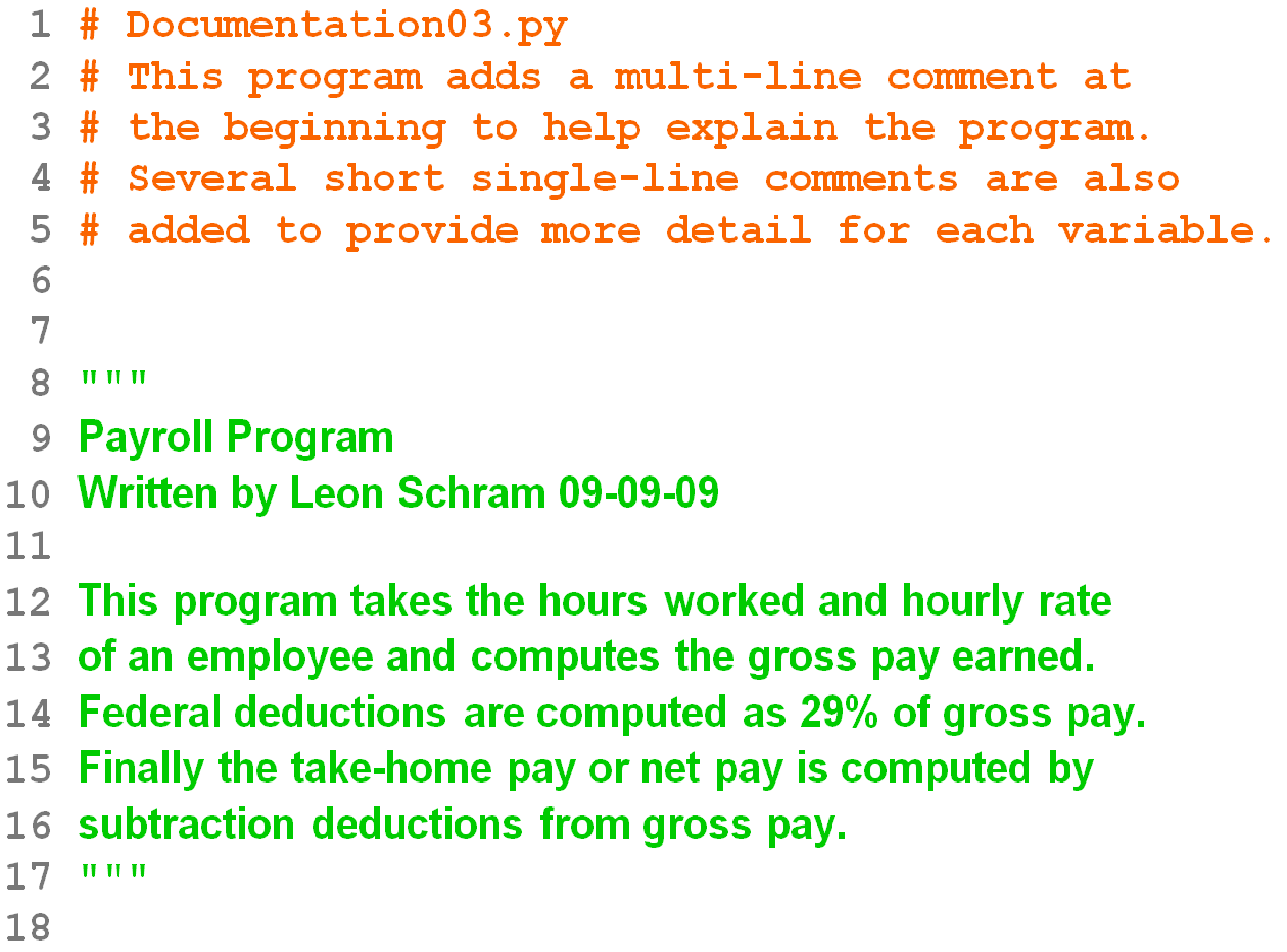
```
 ----jGRASP exec: python

Hours Worked:   35
Hourly Rate:    8.75
Gross Pay:      306.25
Deductions:     88.8125
Net Pay:        217.4375

 ----jGRASP: operation c
```



```python
# Documentation02.py
# This program does ex
# as the previous prog
# documenting variable
# much easier to read


hoursWorked = 35
hourlyRate = 8.75
grossPay = hoursWorked * hourlyRate
deductions = grossPay * 0.29
netPay = grossPay - deductions

print()
print("Hours Worked:  ",hoursWorked)
print("Hourly Rate:   ",hourlyRate)
print("Gross Pay:     ",grossPay)
print("Deductions:    ",deductions)
print("Net Pay:       ",netPay)
```

```
 ----jGRASP exec: python

Hours Worked:   35
Hourly Rate:    8.75
Gross Pay:      306.25
Deductions:     88.8125
Net Pay:        217.4375

 ----jGRASP: operation c
```

NOTE: The output of this program shows dollar amounts that are not rounded to the nearest penny. You will learn how to format your output in a later chapter.



```python
# Documentation03.py
# This program adds a multi-line comment at
# the beginning to help explain the program.
# Several short single-line comments are also
# added to provide more detail for each variable.


"""
Payroll Program
Written by Leon Schram 09-09-09

This program takes the hours worked and hourly rate
of an employee and computes the gross pay earned.
Federal deductions are computed as 29% of gross pay.
Finally the take-home pay or net pay is computed by
subtraction deductions from gross pay.
"""

```

```
hoursWorked = 35   #  hours worked per week
hourlyRate = 8.75  #  pay rate earned per hour
grossPay = hoursWorked * hourlyRate  #  total earnings
deductions = grossPay * 0.29         #  federal tax
netPay = grossPay - deductions       #  take home pay

print()
print("Hours Worked: ",hoursWorked)
print("Hourly Rate:  ",hourlyRate)
print("Gross Pay:    ",grossPay)
print("Deductions:   ",deductions)
print("Net Pay:      ",netPay)
```

This program has the exact same output as the previous program.

2 Types of Comments Review

```
# This is a single-line comment.

hoursWorked = 35    # So is this.
```

```
"""
This is a
multi-line
comment.
"""
```