

# Variables, Expressions, and Statements

Lecture 2

# Constants

- **Fixed values** such as numbers, letters, and strings are called “**constants**” - because their value does not change
- Numeric **constants** are as you expect
- String **constants** use single-quotes (') or double-quotes (")

```
>>> print 123
```

```
123
```

```
>>> print 98.6
```

```
98.6
```

```
>>> print 'Hello world'
```

```
Hello world
```

# Variables

- A **variable** is a named place in the memory where a programmer can store data and later retrieve the data using the **variable** “name”
- Programmers get to choose the names of the **variables**
- You can change the contents of a **variable** in a later statement

**x** = 12.2

**y** = 14

**x** = 100

**x** ~~12.2~~ 100

**y** 14

# Python Variable Name Rules

- Must start with a letter or underscore \_
- Must consist of letters and numbers and underscores
- Case Sensitive
- Good: spam eggs spam23 \_speed
- Bad: 23spam #sign var.12
- Different: spam Spam SPAM

# Reserved Words

- You can not use **reserved words** as variable names / identifiers

and del for is raise  
assert elif from lambda return  
break else global not try  
class except if or while  
continue exec import pass yield  
def finally in print

# Sentences or Lines

`x = 2`

← Assignment Statement

`x = x + 2`

← Assignment with expression

`print x`

← Print statement

Variable

Operator

Constant

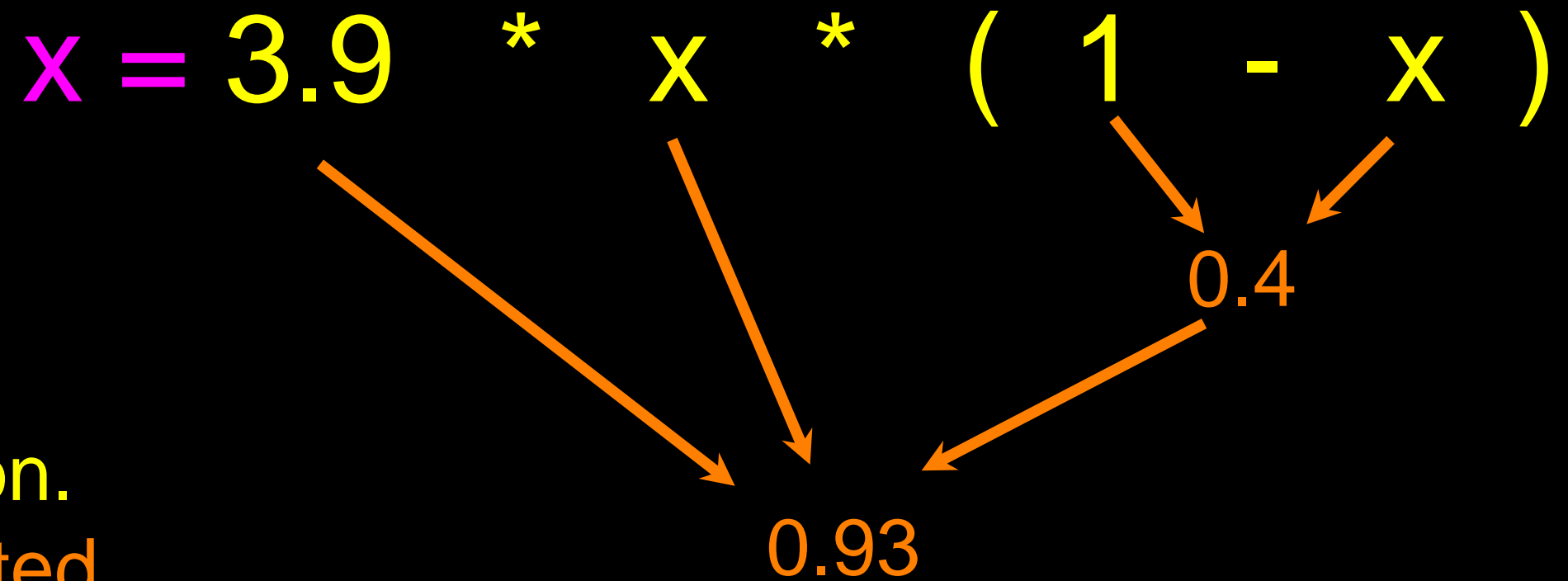
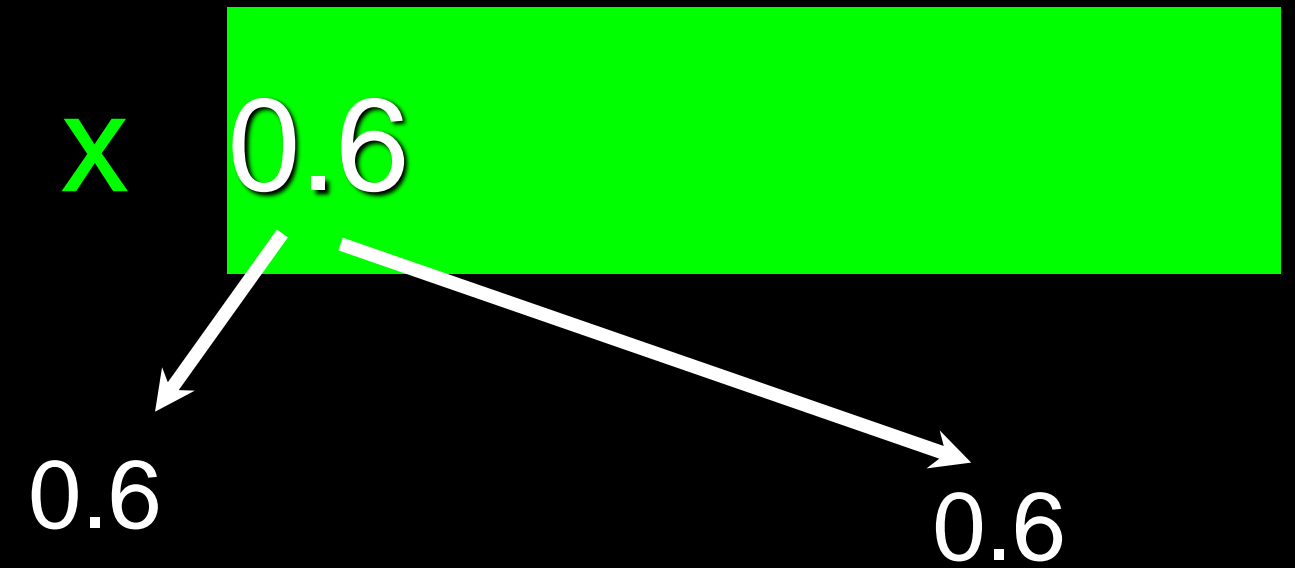
Reserved Word

# Assignment Statements

- We assign a value to a variable using the **assignment** statement (=)
- An **assignment statement** consists of an **expression on the right hand side** and a **variable** to store the result

**x** = 3.9 \* **x** \* ( 1 - **x** )

A variable is a memory location used to store a value (0.6).



Right side is an expression.  
Once expression is evaluated,  
the result is placed in (assigned  
to) `x`.



A variable is a memory location used to store a value. The value stored in a variable can be updated by replacing the old value (0.6) with a new value (0.93).



$$x = 3.9 * x * (1 - x)$$

Right side is an expression.  
Once expression is evaluated, the result is placed in (assigned to) the variable on the left side (i.e.  $x$ ).

0.93

# Numeric Expressions

- Because of the lack of mathematical symbols on computer keyboards - we use “computer-speak” to express the classic math operations
- Asterisk is multiplication
- Exponentiation (raise to a power) looks different from in math.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

# Numeric Expressions

```
>>> xx = 2
>>> xx = xx + 2
>>> print xx
4
>>> yy = 440 * 12
>>> print yy
5280
>>> zz = yy / 1000
>>> print zz
5
```

```
>>> jj = 23
>>> kk = jj % 5
>>> print kk
3
>>> print 4 ** 3
64
```

```
      4 R 3
5 | 23
   20
   --
    3
```

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

# Order of Evaluation

- When we string operators together - Python must know which one to do first
- This is called “operator precedence”
- Which operator “takes precedence” over the others

`x = 1 + 2 * 3 - 4 / 5 ** 6`

# Operator Precedence Rules

- Highest precedence rule to lowest precedence rule
  - Parenthesis are always respected
  - Exponentiation (raise to a power)
  - Multiplication, Division, and Remainder
  - Addition and Subtraction
  - Left to right

Parenthesis  
Power  
Multiplication  
Addition  
Left to Right



```
>>> x = 1 + 2 ** 3 / 4 * 5
```

```
>>> print x
```

```
11
```

```
>>>
```

Parenthesis

Power

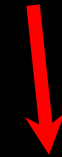
Multiplication

Addition

Left to Right



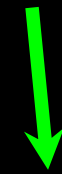
1 + 2 \*\* 3 / 4 \* 5



1 + 8 / 4 \* 5



1 + 2 \* 5



1 + 10



11

```
>>> x = 1 + 2 ** 3 / 4 * 5
```

```
>>> print x
```

```
11
```

```
>>>
```

Note 8/4 goes before 4\*5  
because of the left-right  
rule.

$$1 + 2 ** 3 / 4 * 5$$

$$1 + 8 / 4 * 5$$

$$1 + 2 * 5$$

$$1 + 10$$

11

Parenthesis

Power

Multiplication

Addition

Left to Right



# Operator Precedence

Parenthesis  
Power  
Multiplication  
Addition  
Left to Right



- Remember the rules top to bottom
- When writing code - use parenthesis
- When writing code - keep mathematical expressions simple enough that they are easy to understand
- Break long series of mathematical operations up to make them more clear

Exam Question:  $x = 1 + 2 * 3 - 4 / 5$



# Python Integer Division is Weird!

- Integer division truncates
- Floating point division produces floating point numbers

```
>>> print 10 / 2  
5
```

```
>>> print 9 / 2  
4
```

```
>>> print 99 / 100  
0
```

```
>>> print 10.0 / 2.0  
5.0
```

```
>>> print 99.0 / 100.0  
0.99
```

This changes in Python 3.0

# Mixing Integer and Floating

- When you perform an operation where one operand is an integer and the other operand is a floating point the result is a floating point
- The integer is converted to a floating point before the operation

```
>>> print 99 / 100
```

```
0
```

```
>>> print 99 / 100.0
```

```
0.99
```

```
>>> print 99.0 / 100
```

```
0.99
```

```
>>> print 1 + 2 * 3 / 4.0 - 5
```

```
-2.5
```

```
>>>
```

# What does “Type” Mean?

- In Python variables, literals, and constants have a “type”
- Python knows the difference between an integer number and a string
- For example “+” means “addition” if something is a number and “concatenate” if something is a string

```
>>> ddd = 1 + 4
```

```
>>> print ddd
```

```
5
```

```
>>> eee = 'hello ' + 'there'
```

```
>>> print eee
```

```
hello there
```

concatenate = put together

# Type Matters

- Python knows what “type” everything is
- Some operations are prohibited
- You cannot “add 1” to a string
- We can ask Python what type something is by using the `type()` function.

```
>>> eee = 'hello ' + 'there'
```

```
>>> eee = eee + 1
```

```
Traceback (most recent call  
last):
```

```
  File "<stdin>", line 1, in  
<module>
```

```
TypeError: cannot concatenate  
'str' and 'int' objects
```

```
>>> type(eee)
```

```
<type 'str'>
```

```
>>> type('hello')
```

```
<type 'str'>
```

```
>>> type(1)
```

```
<type 'int'>
```

```
>>>
```

# Several **Types** of Numbers

- Numbers have two main types
  - Integers are whole numbers: -14, -2, 0, 1, 100, 401233
  - Floating Point Numbers have decimal parts: -2.5 , 0.0, 98.6, 14.0
- There are other number types - they are variations on float and integer

```
>>> xx = 1
>>> type (xx)
<type 'int'>
>>> temp = 98.6
>>> type(temp)
<type 'float'>
>>> type(1)
<type 'int'>
>>> type(1.0)
<type 'float'>
>>>
```

# Type Conversions

- When you put an integer and floating point in an expression the integer is **implicitly** converted to a float
- You can control this with the built in functions `int()` and `float()`

```
>>> print float(99) / 100  
0.99
```

```
>>> i = 42
```

```
>>> type(i)  
<type 'int'>
```

```
>>> f = float(i)
```

```
>>> print f  
42.0
```

```
>>> type(f)  
<type 'float'>
```

```
>>> print 1 + 2 * float(3) / 4 - 5  
-2.5
```

```
>>>
```

# String Conversions

- You can also use `int()` and `float()` to convert between strings and integers
- You will get an **error** if the string does not contain numeric characters

```
>>> sval = '123'
```

```
>>> type(sval)
```

```
<type 'str'>
```

```
>>> print sval + 1
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: cannot concatenate 'str' and 'int'
```

```
>>> ival = int(sval)
```

```
>>> type(ival)
```

```
<type 'int'>
```

```
>>> print ival + 1
```

```
124
```

```
>>> nsv = 'hello bob'
```

```
>>> niv = int(nsv)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
ValueError: invalid literal for int()
```

# User Input

- We can instruct Python to pause and read data from the user using the `raw_input` function
- The `raw_input` function returns a string

```
nam = raw_input( 'Who are you?'  
print 'Welcome', nam
```

```
Who are you? Chuck  
Welcome Chuck
```



# Converting User Input



- If we want to read a number from the user, we must convert it from a string to a number using a type conversion function
- Later we will deal with bad input data

```
inp = raw_input( 'Europe floor?' )  
usf = int(inp) + 1  
print 'US floor', usf
```

Europe floor? 0  
US floor 1

# Comments in Python

- Anything after a # is ignored by Python
- Why comment?
  - Describe what is going to happen in a sequence of code
  - Document who wrote the code or other ancillary information
  - Turn off a line of code - perhaps temporarily

**# Get the name of the file and open it**

```
name = raw_input('Enter file:')
```

```
handle = open(name, 'r')
```

```
text = handle.read()
```

```
words = text.split()
```

**# Count word frequency**

```
counts = dict()
```

```
for word in words:
```

```
    counts[word] = counts.get(word,0) + 1
```

**# Find the most common word**

```
bigcount = None
```

```
bigword = None
```

```
for word,count in counts.items():
```

```
    if bigcount is None or count > bigcount:
```

```
        bigword = word
```

```
        bigcount = count
```

**# All done**

```
print bigword, bigcount
```

# String Operations

- Some operators apply to strings
  - `+` implies “concatenation”
  - `*` implies “multiple concatenation”
- Python knows when it is dealing with a string or a number and behaves appropriately

```
>>> print 'abc' + '123'  
Abc123  
>>> print 'Hi' * 5  
HiHiHiHiHi  
>>>
```

# Mnemonic Variable Names

- Since we programmers are given a choice in how we choose our variable names, there is a bit of “best practice”
- We name variables to help us remember what we intend to store in them (“mnemonic” = “memory aid”)
- This can confuse beginning students because well named variables often “sound” so good that they must be keywords

<http://en.wikipedia.org/wiki/Mnemonic>

```
x1q3z9ocd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ocd * x1q3z9afd
print x1q3p9afd
```

```
a = 35.0
b = 12.50
c = a * b
print c
```

What is this  
code doing?

```
hours = 35.0
rate = 12.50
pay = hours * rate
print pay
```

## Exercise

Write a program to prompt the user for hours and rate per hour to compute gross pay.

Enter Hours: 35

Enter Rate: 2.75

Pay: 96.25

# Summary

- Type
- Resrved words
- Variables (mnemonic)
- Operators
- Operator precedence
- Integer Division
- Conversion between types
- User input
- Comments (#)