Python Programming

LECTURES BY

DR. AVINASH GULVE

Lecture -2

- Working with Python
- Constants
- Variables

Programming Languages

A *program* is just a sequence of instructions telling the computer what to do

Obviously, we need to provide these instructions in a language that computers can understand

- We refer to this kind of a language as a *programming language*
- Python, Java, C and C++ are examples of programming languages

Every structure in a programming language has an exact form (i.e., *syntax*) and a precise meaning (i.e., *semantic*)

Machine Languages

Python, Java, C, and C++ are, indeed, examples of *high-level* languages

Strictly speaking, computer hardware can only understand a very *low-level* language known as *machine language*

If you want a computer to add two numbers, the instructions that the CPU will carry out might be something like this:

Load the number from memory location 2001 into the CPU
Load the number from memory location 2002 into the CPU
Add the two numbers in the CPU
Store the result into location 2003

A Lot of Work!

High-Level to Low-Level Languages

In a high-level language like Python, the addition of two numbers can be expressed more naturally:

```
c = a + b
```

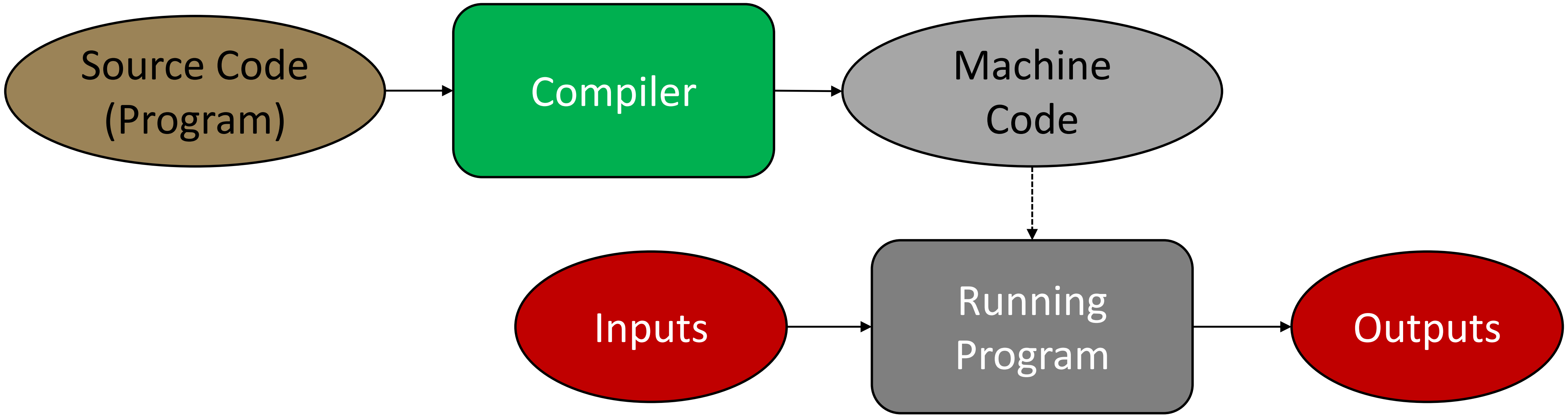
Much Easier!

But, we need a way to translate the high-level language into a machine language that a computer can execute

- To this end, high-level language can either be *compiled* or *interpreted*

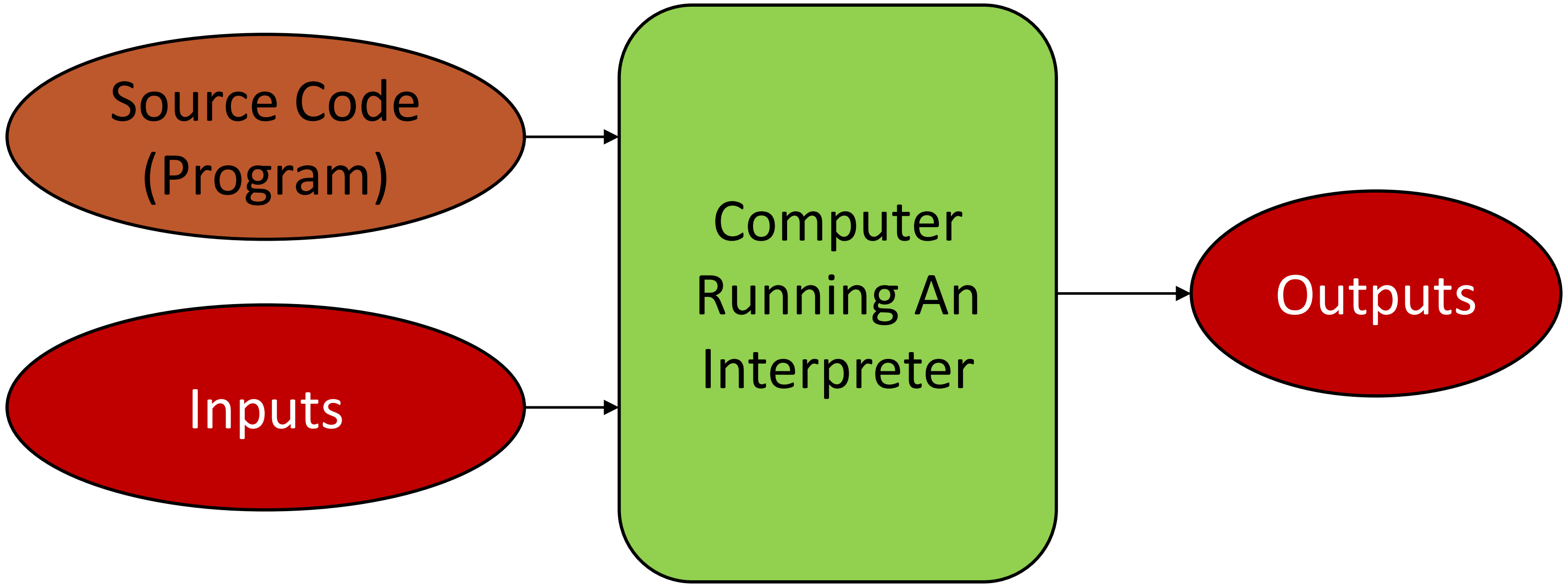
Compiling a High-Level Language

A *compiler* is a complex software that takes a program written in a high-level language and *translates* it into an equivalent program in the machine language of some computer



Interpreting a High-Level Language

An *interpreter* is a software that analyzes and executes the source code instruction-by-instruction (*on-the-fly*) as necessary



E.g., Python is an interpreted language

Programming in Script Mode

Interactive mode gives you immediate feedback

Not designed to create programs to save and run later

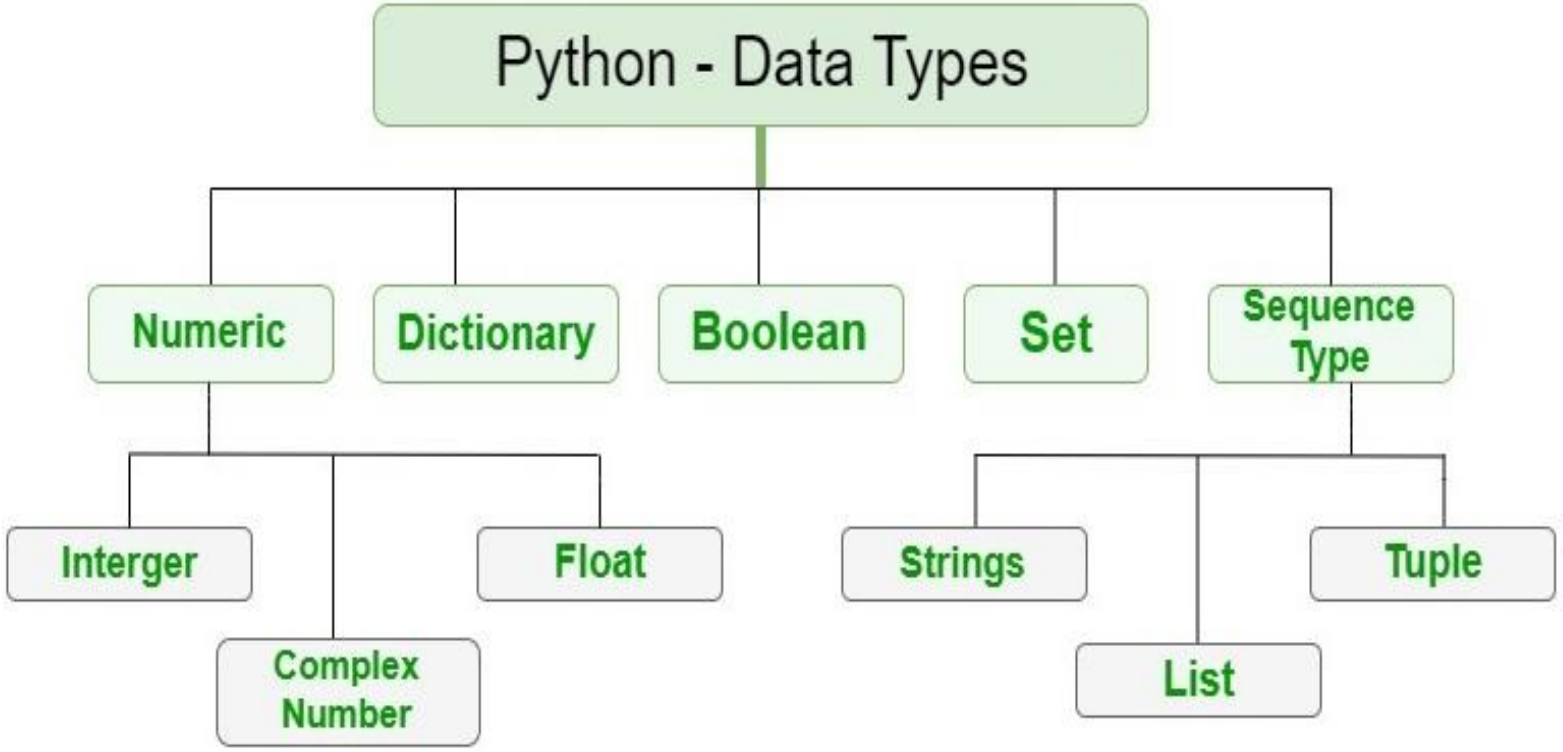
Script Mode

- Write, edit, save, and run (later)
  - Word processor for your code

Save your file using the “.py” extension

Data Types

- Numeric
- Sequence Type
- Boolean
- Set
- Dictionary



Data Types

Number - Numeric data type represent the data which has numeric value. Numeric value can be integer, floating number or even complex numbers.

Sequence Type - Sequence is the ordered collection of similar or different data types. Sequences allows to store multiple values in an organized and efficient fashion.

1. Strings are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. There is no character data type, a character is a string of length one.

2. Lists are just like the arrays, declared in other languages which is a ordered collection of data. It is very flexible as the items in a list do not need to be of the same type. Created using []

3. Tuple is also an ordered collection of Python objects. The only difference between type and list is that tuples are immutable i.e. tuples cannot be modified after it is created. Created using ()

Data Types

Boolean-	Data type with one of the two built-in values, True or False.
Set-	Set is an unordered collection of data type that is iterable, mutable and has no duplicate elements. Created using {}
Dictionary-	Dictionary is an unordered collection of data values. Dictionary holds key:value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon :, whereas each key is separated by a 'comma'.
	Dictionaries are written with {} and have keys and values

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
```

Literals

- In the following example, the parameter values passed to the print function are all technically called *literals*
  - More precisely, “Hello” and “Programming is fun!” are called *textual literals*, while 3 and 2.3 are called *numeric literals*

```
>>> print("Hello")
Hello
>>> print("Programming is fun!")
Programming is fun!
>>> print(3)
3
>>> print(2.3)
2.3
```

Identifiers

- Python has some rules about how identifiers can be formed
  - Every identifier must begin with a letter or underscore, which may be followed by any sequence of letters, digits, or underscores
  - Identifiers are *case-sensitive*

```
>>> x = 10
>>> X = 5.7
>>> print(x)
10
>>> print(X)
5.7
```

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

Program Documentation

All code must contain comments that describe what it does

In Python, lines beginning with a # sign are comment lines

- Ignored by the computer

```
# Programmer
# First Python Program
# September 1, 2005
```

Variables

No need to declare

Need to assign (initialize)

- use of uninitialized variable raises exception

Not typed

```
if friendly: greeting = "hello world"
else: greeting = 12**2
print greeting
```

Everything is a "variable":

- Even functions, classes, modules

Variables

Assignment manipulates references

- x = y **does not make a copy** of y
- x = y makes x **reference** the object y references

Name that represents a value stored in computer memory

- Used to remember, access and manipulate data stored in memory
- Variable references the value

Assignment statement - used to create a variable and make it reference the desired data

- General format is
  - variable = expression
  - Expression is the value, mathematical equation or function that results in a value
  - Assignment operator: the equal sign (=)

Variables

- In assignment statement, variable receiving value must be on left side
- You cannot use a variable UNTIL a value is assigned to it
  - Otherwise will "get name 'variable name' is not defined"
- A variable can be passed as an argument to a function, like print
  - Do not enclose variable name in quotes
  - Examples:

•Good:	spam eggs spam23 _speed
•Bad:	23spam #sign var.12
•Different:	spam Spam SPAM

Variable Naming Rules

Rules for naming variables in Python

- Should be short, but descriptive
- First character must be a letter or an underscore
- After first character may use letters, digits, or underscores
- Variable name cannot be a Python keyword
- Variable name cannot contain spaces
  - Use underscore instead
- Variable names are case sensitive

Variable name should reflect its use

- Ex: grosspay, payrate
- Separate words: gross_pay, pay_rate

Example Program

```
message = "Hello Python world!"

print(message)
```

Output:

```
Hello Python world!
```

Change Variable Value

```
message = "Hello Python world!"

print(message)

message = "Hello world!” # message has new value

print(message)         # something different is printed
```

Output:

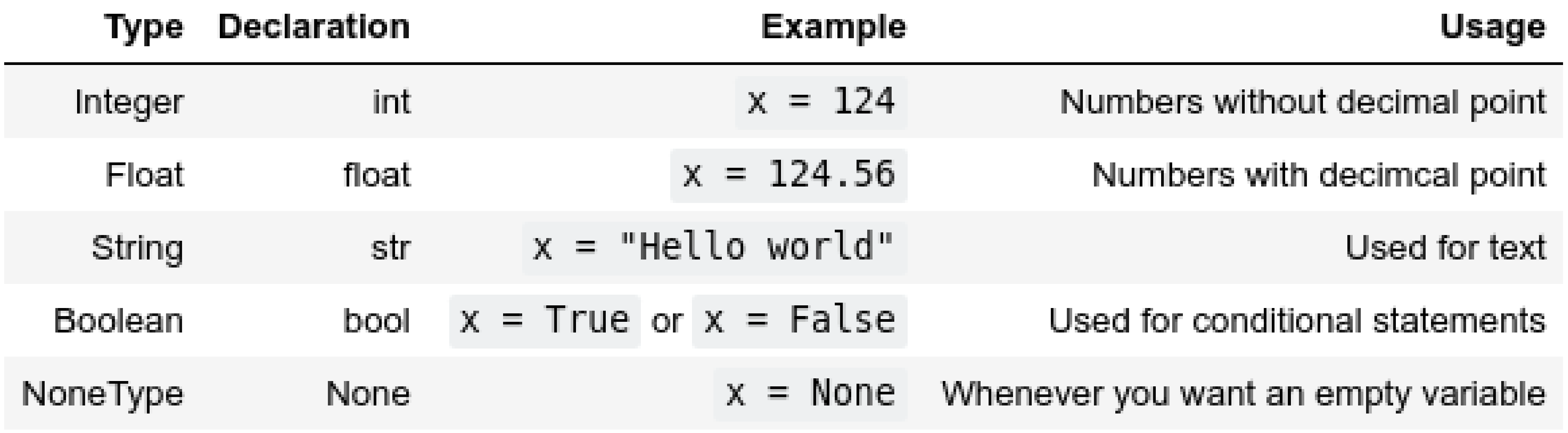
```
Hello Python world!

Hello world!
```

Variables

Variables actually have a type, which defines the way it is stored.

The basic types are: Numbers, Strings, Boolean and None



Type	Declaration	Example	Usage
Integer	int	`x = 124`	Numbers without decimal point
Float	float	`x = 124.56`	Numbers with decimcal point
String	str	`x = "Hello world"`	Used for text
Boolean	bool	`x = True` or `x = False`	Used for conditional statements
NoneType	None	`x = None`	Whenever you want an empty variable

Simple Assignment Statements

A literal is used to indicate a specific value, which can be *assigned* to a *variable*

- x is a variable and 2 is its value

```
>>> x = 2
>>> print(x)
2
>>> x = 2.3
>>> print(x)
2.3
```

Simple Assignment Statements

A literal is used to indicate a specific value, which can be *assigned* to a *variable*

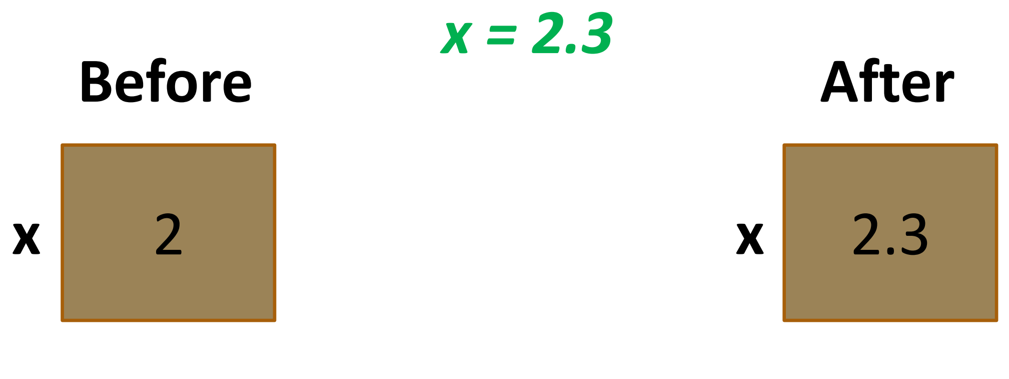
- x is a variable and 2 is its value
- x can be assigned different values; hence, it is called a variable

```
>>> x = 2
>>> print(x)
2
>>> x = 2.3
>>> print(x)
2.3
```

Simple Assignment Statements: Box View

A simple way to view the effect of an assignment is to assume that when a variable changes, its old value is replaced

```
>>> x = 2
>>> print(x)
2
>>> x = 2.3
>>> print(x)
2.3
```

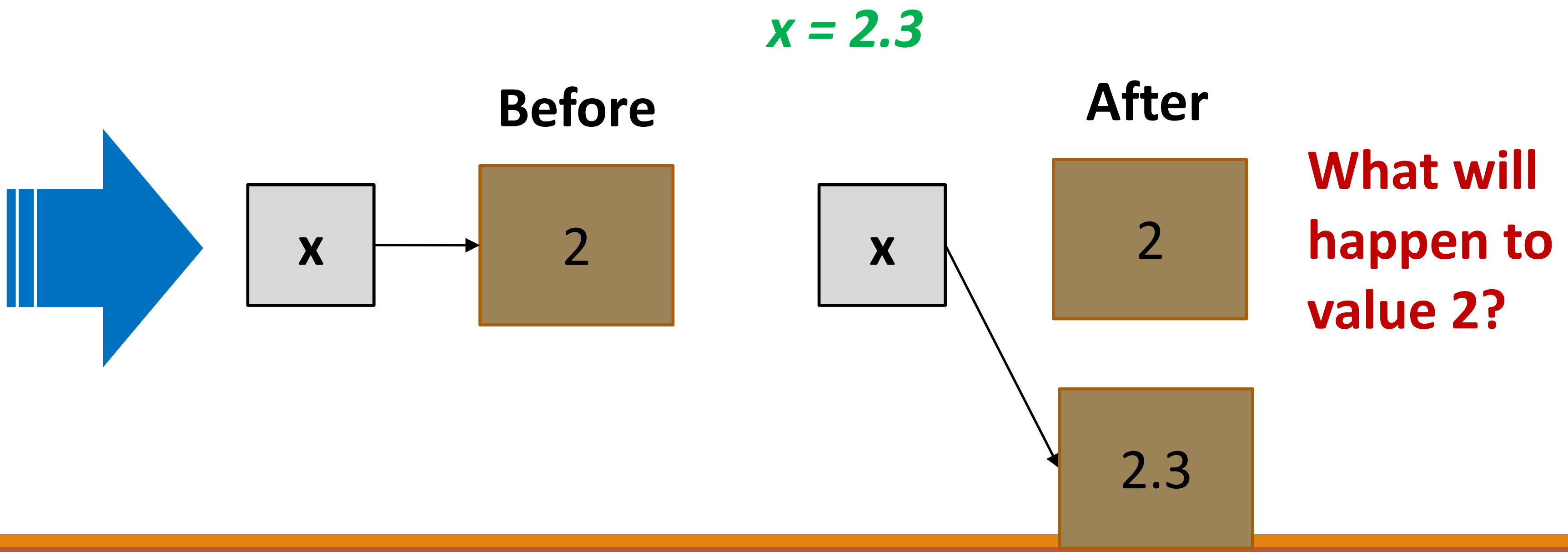


Simple Assignment Statements: Actual View

Python assignment statements are actually slightly different from the “variable as a box” model

- In Python, values may end up anywhere in memory, and variables are used to refer to them

```
>>> x = 2
>>> print(x)
2
>>> x = 2.3
>>> print(x)
2.3
```

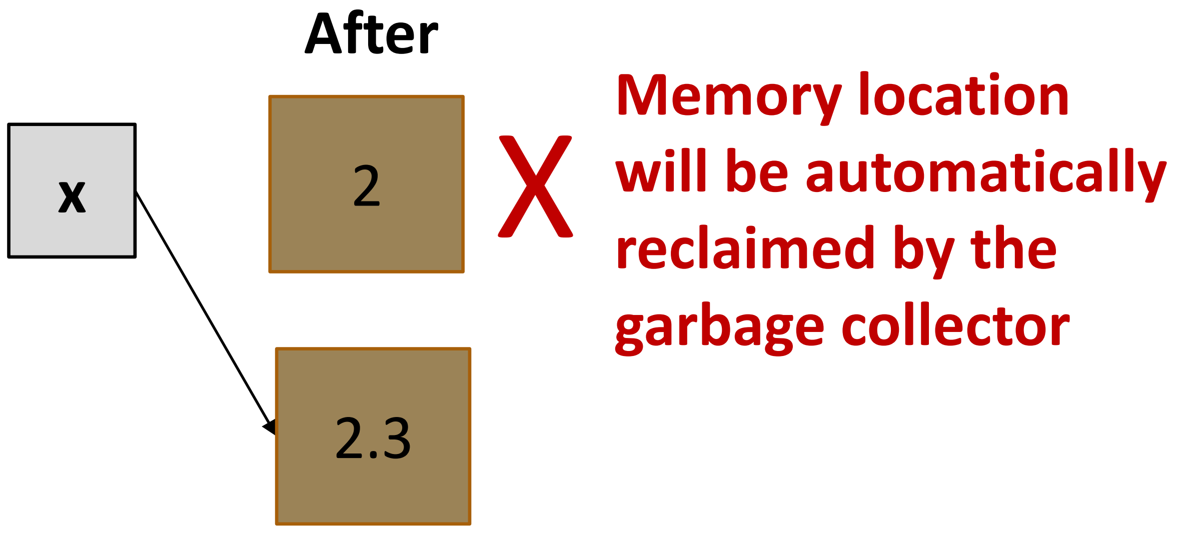


Garbage Collection

Previous value may still stored in memory but lost the handle to it.

Interestingly, as a Python programmer you do not have to worry about computer memory getting filled up with old values when new values are assigned to variables

Python will automatically clear old values out of memory in a process known as *garbage collection*


After
x
2
X
2.3
Memory location will be automatically reclaimed by the garbage collector

Type

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

Type

```
>>> type('hello')
<type 'str'>
>>> type(1)
<type 'int'>
```

Variables and Scope

- Module: one python file.
- Global scope: exists in the module in which they are declared.
- Local scope: local to the function inside which it is declared.
- Global variables in a module can be accessed from somewhere else

Datatype Conversion

Besides, we can convert the string output of the *input* function into an integer or a float using the built-in ***int*** and ***float*** functions

```
>>> number = float(input("Enter a number: "))
Enter a number: 3.7
>>> number
3.7
>>>
```

A float (no single quotes)!

Datatype Conversion

- As a matter of fact, we can do various kinds of conversions between strings, integers and floats using the built-in ***int***, ***float***, and ***str*** functions

```
>>> x = 10
>>> float(x)
10.0
>>> str(x)
'10'
>>>
```

integer → float
integer → string

```
>>> y = "20"
>>> float(y)
20.0
>>> int(y)
20
>>>
```

string → float
string → integer

```
>>> z = 30.0
>>> int(z)
30
>>> str(z)
'30.0'
>>>
```

float → integer
float → string

Explicit and Implicit Data Type Conversion

Data conversion can happen in two ways in Python

1. Explicit Data Conversion (we saw this earlier with the *int*, *float*, and *str* built-in functions)

2. Implicit Data Conversion

- Takes place *automatically* during run time between *ONLY* numeric values
  - E.g., Adding a float and an integer will automatically result in a float value
  - E.g., Adding a string and an integer (or a float) will result in an *error* since string is not numeric
- Applies *type promotion* to avoid loss of information
  - Conversion goes from integer to float (e.g., upon adding a float and an integer) and not vice versa so as the fractional part of the float is not lost

Implicit Data Type Conversion: Examples

- The result of an expression that involves a float number alongside (an) integer number(s) is a float number

```
>>> print(2 + 3.4)
5.4
>>> print( 2 + 3)
5
>>> print(9/5 * 27 + 32)
80.6
>>> print(9//5 * 27 + 32)
59
>>> print(5.9 + 4.2)
10.100000000000001
>>>
```

Implicit Data Type Conversion: Examples

- The result of an expression that involves a float number alongside (an) integer number(s) is a float number
- The result of an expression that involves values of the same data type will not result in any conversion

```
>>> print(2 + 3.4)
5.4
>>> print( 2 + 3)
5
>>> print(9/5 * 27 + 32)
80.6
>>> print(9//5 * 27 + 32)
59
>>> print(5.9 + 4.2)
10.100000000000001
>>>
```