Introduction to Software Technology

Unit 4: Python

School of computer science

Wuhan University

4.1 Overview of Python

- **Introduction**
- **Downloading and installing python**
- **Running python programs**

Python!

- Created in 1991 by Guido van Rossum (now at Google)
  - Named for Monty Python

- Useful as a **scripting language**
  - **script**: A small program meant for one-time use
  - Targeted towards small to medium sized projects

- Used by:
  - Google, Yahoo!, Youtube
  - Many Linux distributions
  - Games and apps (e.g. Eve Online)

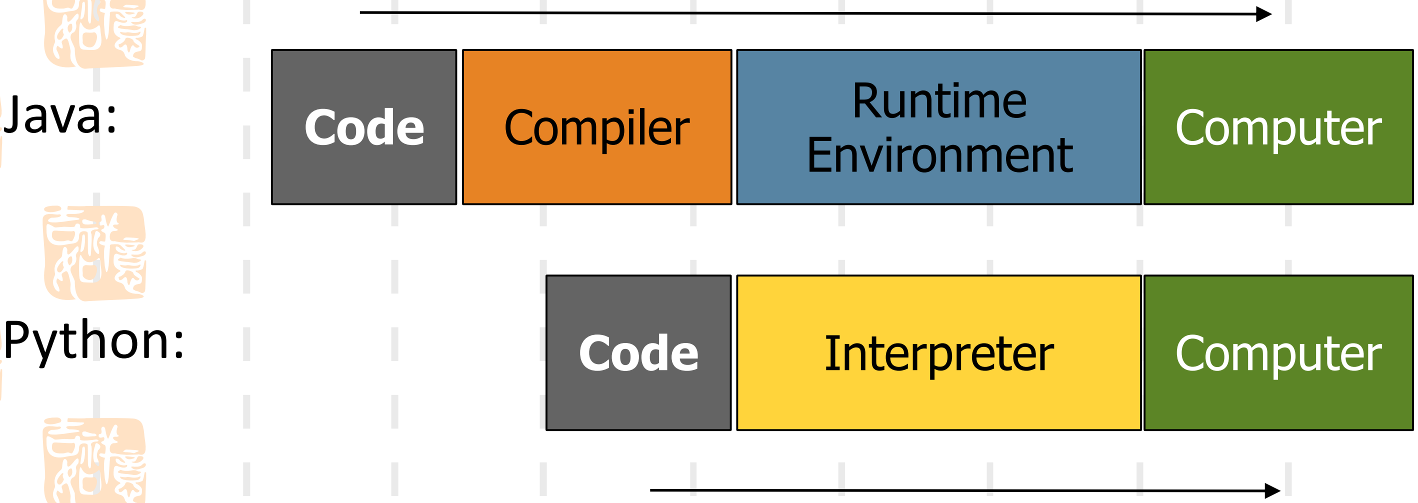
  - Life is short, you need python!

Python is used everywhere!

BitTorrent™
NASA
LUCASFILM Ltd
Google
YouTube
SID MEIER'S CIVILIZATION V

Interpreted Languages

- **Interpreted**
  - Not compiled like Java
  - Code is written and then directly executed by an **interpreter**
  - Type commands into interpreter and see immediate results



Java:

Code | Compiler | Runtime Environment | Computer

Python:

Code | Interpreter | Computer

Interpreter

- An alternative to a compiler is a program called an *interpreter*. Rather than convert our program to the language of the computer, the interpreter takes our program one statement at a time and executes a corresponding set of machine instructions.

Advantages of Python

- Python uses an interpreter. Not only can we write complete programs, we can work with the interpreter in a statement by statement mode enabling us to experiment quite easily.

- Python is especially good for our purposes in that it does not have a lot of “overhead” before getting started.

- It is easy to jump in and experiment with Python in an interactive fashion.

Java vs. Python

- Console output: `System.out.println();`
- Methods: `public static void` **`name`**`() { ... }`

Hello2.java

```java
public class Hello2 {
    public static void main(String[] args) {
        hello();
    }

    public static void hello() {
        System.out.println("Hello, world!");
    }
}
```

Our First Python Program

- Python does not have a `main` method like Java
    - The program's main code is just written directly in the file
- Python statements do not end with semicolons

hello.py

```
print("Hello, world!")
```

4 Major Versions of Python

- “Python” or “CPython” is written in C/C++
- “Jython” is written in Java for the JVM
- “IronPython” is written in C# for the .Net environment
- “PyPy” is written in Python(rPython)

Python 2.x vs Python 3.x

- We will be using Python 3 for this course
- The differences are minimal

How to	Python 2.x	Python 3.x
print text	`print “text”`	`print(“text”)`
print a blank line	`print`	`print()`

Development Environments

what IDE to use?

IDLE

PythonWin

PyCharm

wingIDE

PyDev+Eclipse

Eric

Jupyter Notebook

Spyder

……

Installing Python

Windows:

- Download Python from http://www.python.org
- Install Python.
- Run **Idle** from the Start Menu.

Note: For step by step installation instructions, see the official web site.

Mac OS X:

- Python is already installed.
- Open a terminal and run `python` or run Idle from Finder.

Linux:

- Chances are you already have Python installed. To check, run `python` from the terminal.
- If not, install from your distribution's package system.

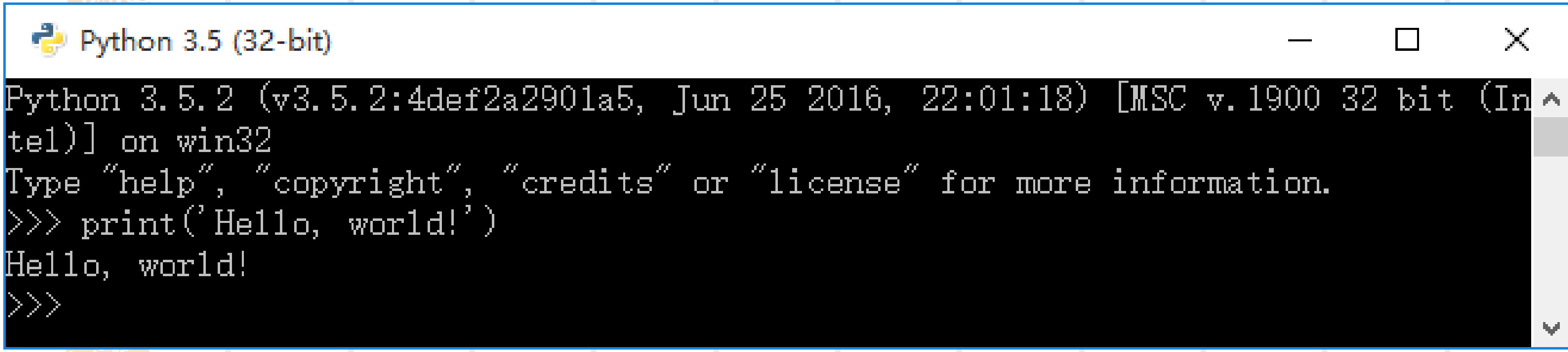
Install and manage Python packages

- pip and setuptools modules are included in Python 3.4 and above
  - Use pip for package installation
  - Use setuptools for Distribution

Run Python program

- Default Installation path for Python

  …AppDat\Local\Python\Python35-32\，include
  - Python interpreter python.exe
  - Python library and other files
- **Run** Python Interpreter
  - Start |Python 3.5|Python 3.5 (32-bit)

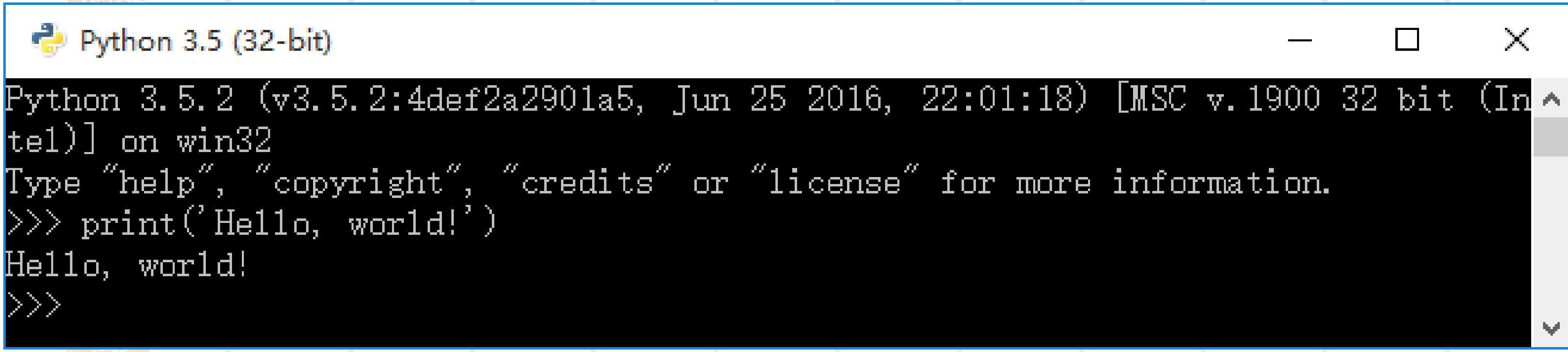


```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello, world!')
Hello, world!
>>>
```

Run Python program

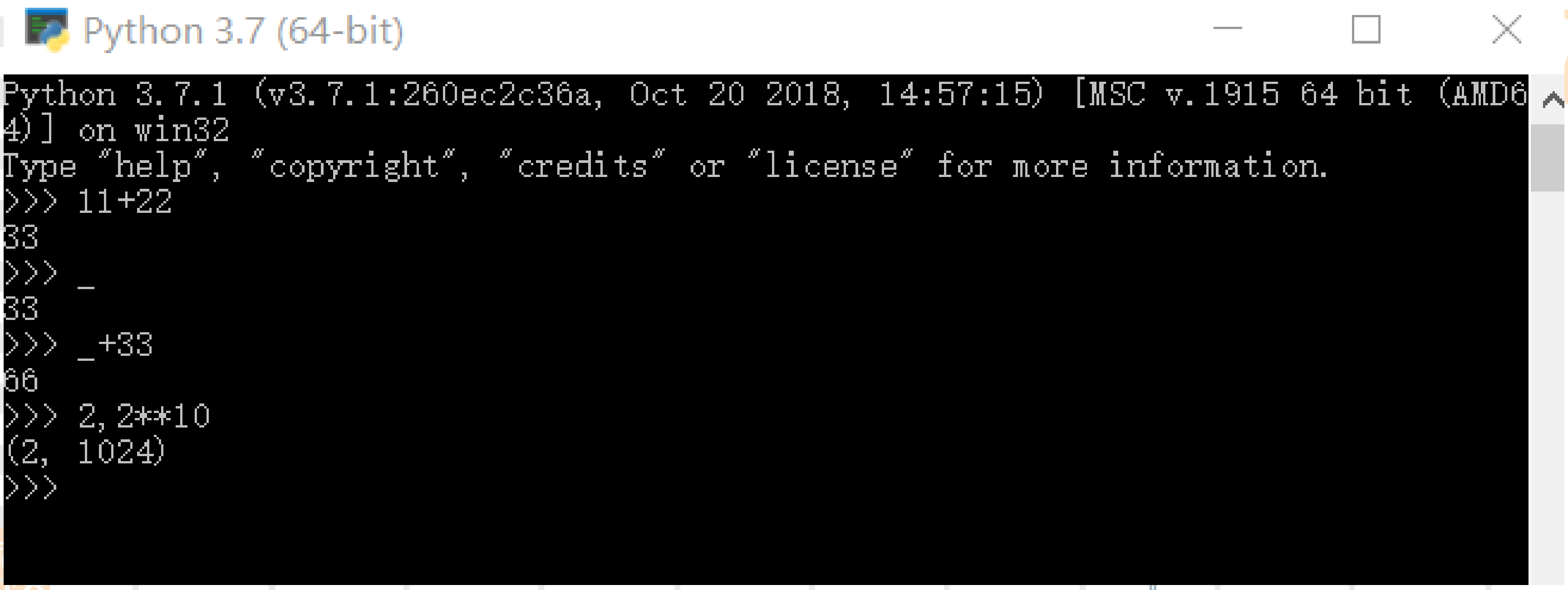
- Default Installation path for Python

  …AppDat\Local\Python\Python35-32\，include
  - Python interpreter python.exe
  - Python library and other files
- **Run** Python Interpreter
  - Start |Python 3.5|Python 3.5 (32-bit)



```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello, world!')
Hello, world!
>>>
```

Run Python Program

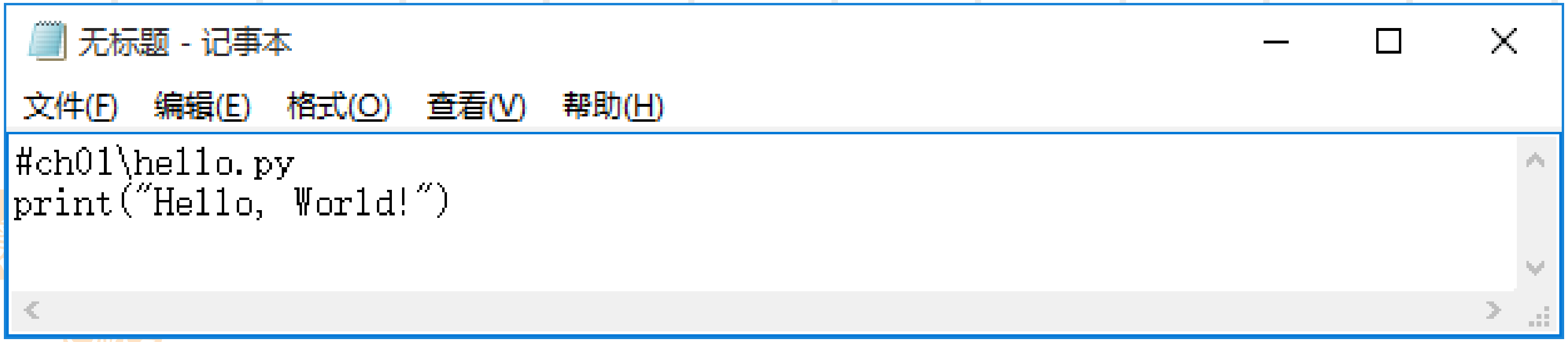


```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [MSC v.1915 64 bit (AMD6
4)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 11+22
33
>>> _
33
>>> _+33
66
>>> 2,2**10
(2, 1024)
>>>
```

- Close Python Interpreter
  - Ctrl+Z
  - quit()
  - Close the window

Edit Python Code and Run

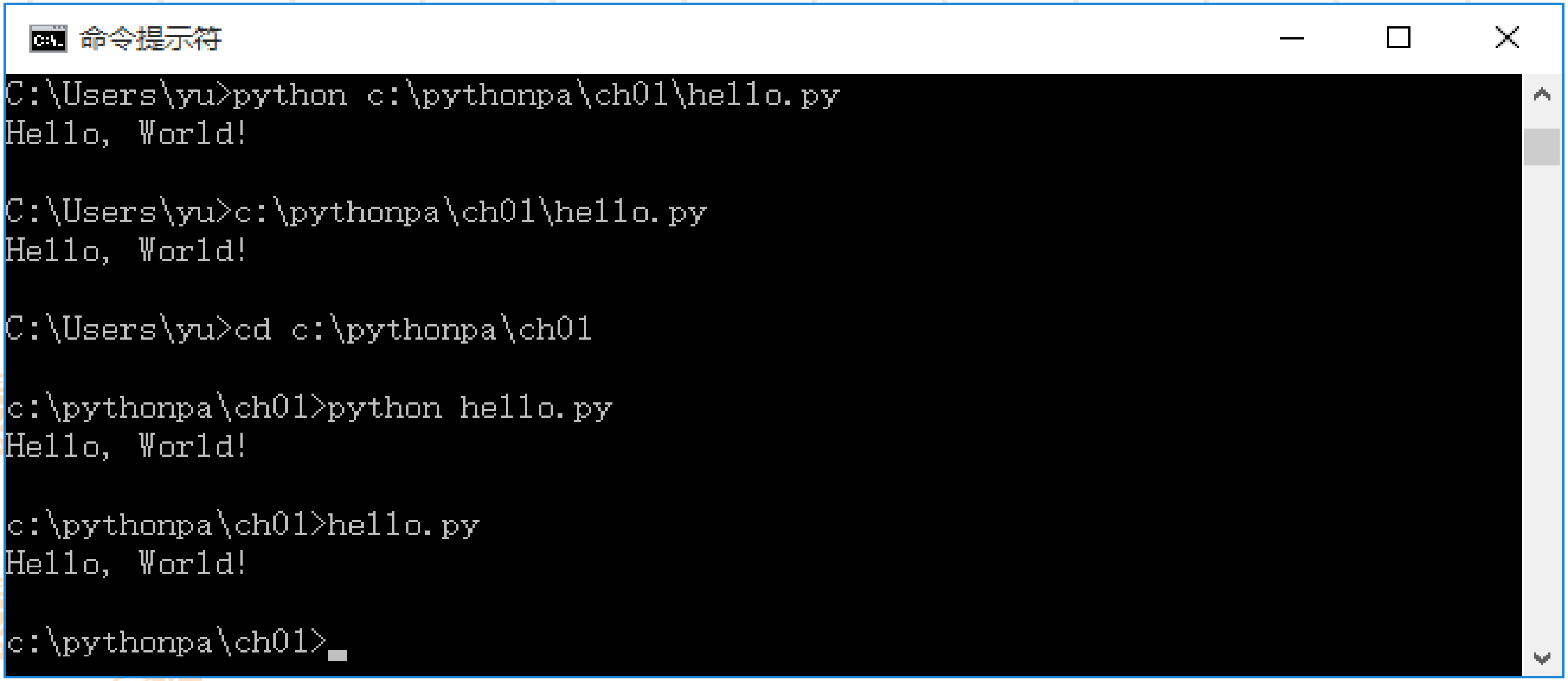
- Use notepad to write python program



```
#ch01\hello.py
print(“Hello, World!”)
```

Run python file in cmd

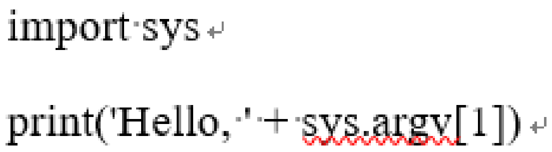
- **Run** hello.py in cmd



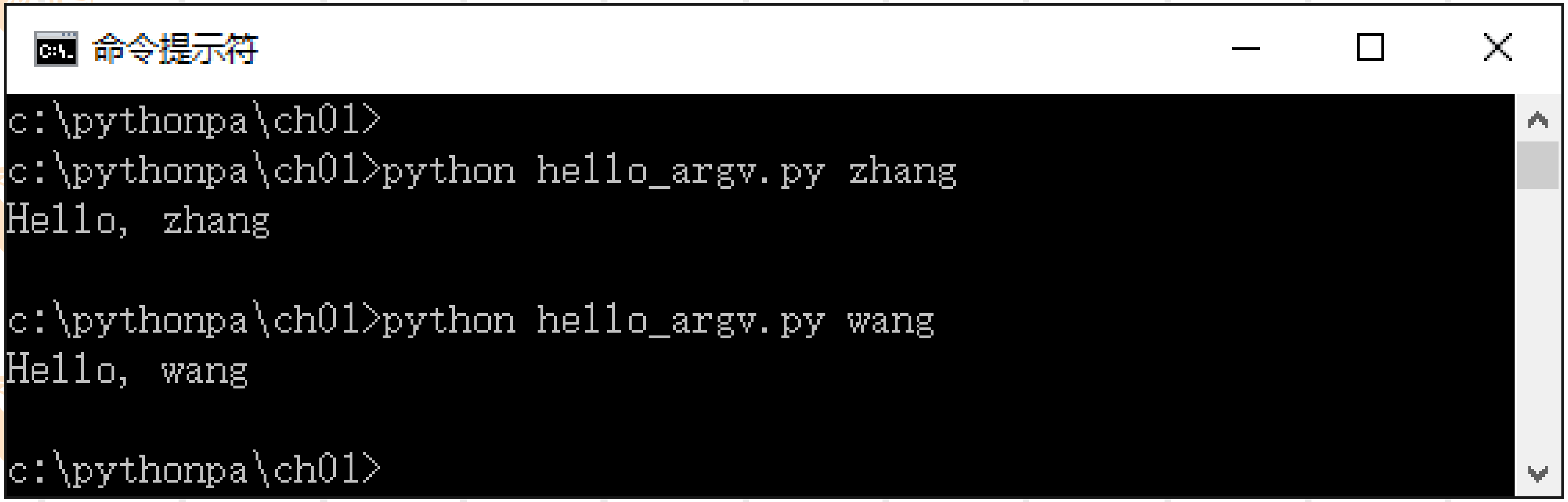
```
C:\Users\yu>python c:\pythonpa\ch01\hello.py
Hello, World!

C:\Users\yu>c:\pythonpa\ch01\hello.py
Hello, World!

C:\Users\yu>cd c:\pythonpa\ch01

c:\pythonpa\ch01>python hello.py
Hello, World!

c:\pythonpa\ch01>hello.py
Hello, World!

c:\pythonpa\ch01>_
```

Run python program with paramters

- hello_argv.py
- Use sys.argv to input paramters.
- The file name is the first parameter. So the input parameter started from argv[1]



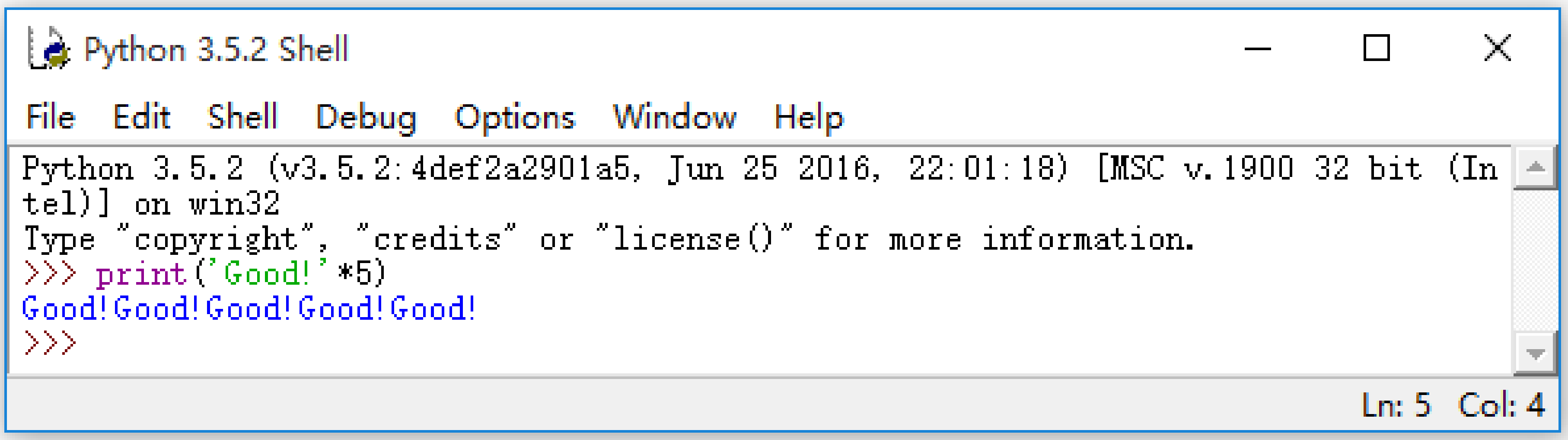
```
import sys

print('Hello, ' + sys.argv[1])
```



```
c:\pythonpa\ch01>
c:\pythonpa\ch01>python hello_argv.py zhang
Hello, zhang

c:\pythonpa\ch01>python hello_argv.py wang
Hello, wang

c:\pythonpa\ch01>
```

Use Python IDLE

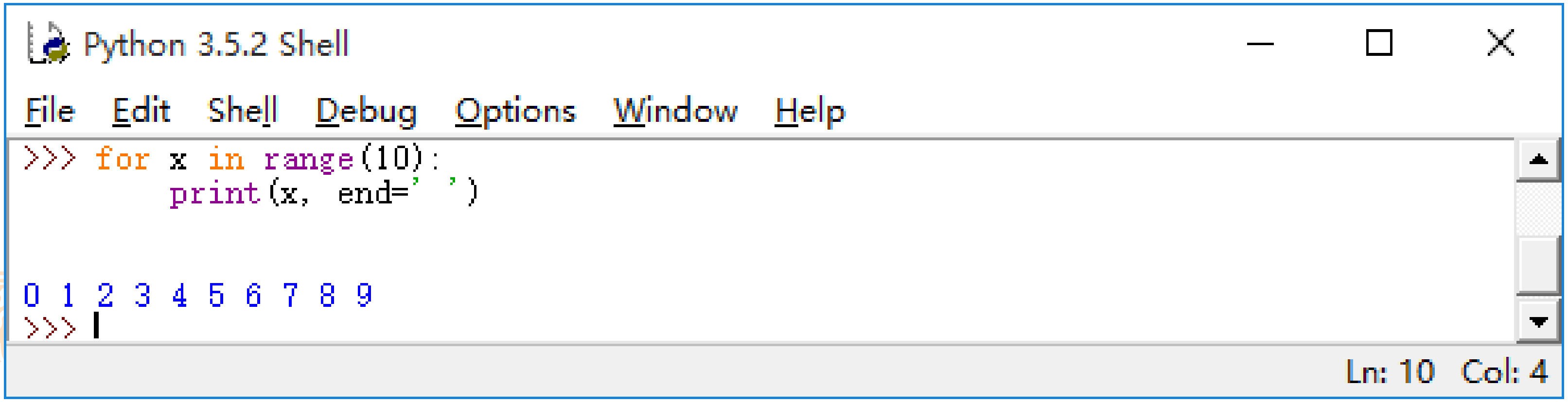
- **Starte** IDLE
  - start| Python 3.5| IDLE (Python 3.5 32-bit)



```
Python 3.5.2 (v3.5.2:4def2a2901a5, Jun 25 2016, 22:01:18) [MSC v.1900 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print('Good!'*5)
Good!Good!Good!Good!Good!
>>>
```

Run IDLE

- **Use IDLE to execute multiple lines.**
  - Print numbers 0-9

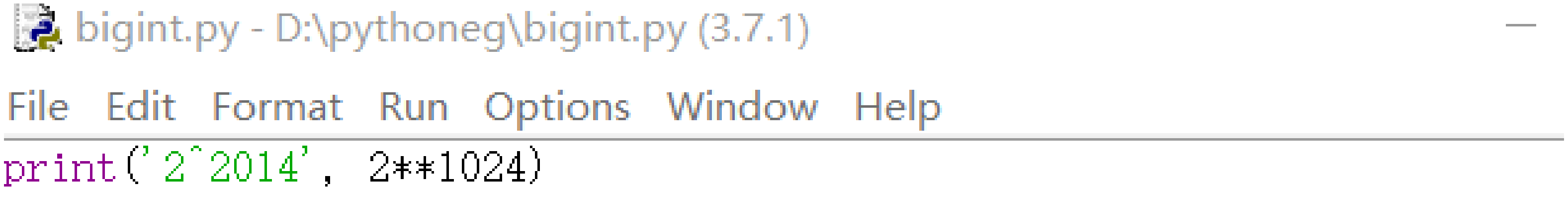


```
>>> for x in range(10):
        print(x, end=' ')


0 1 2 3 4 5 6 7 8 9
>>> 
```

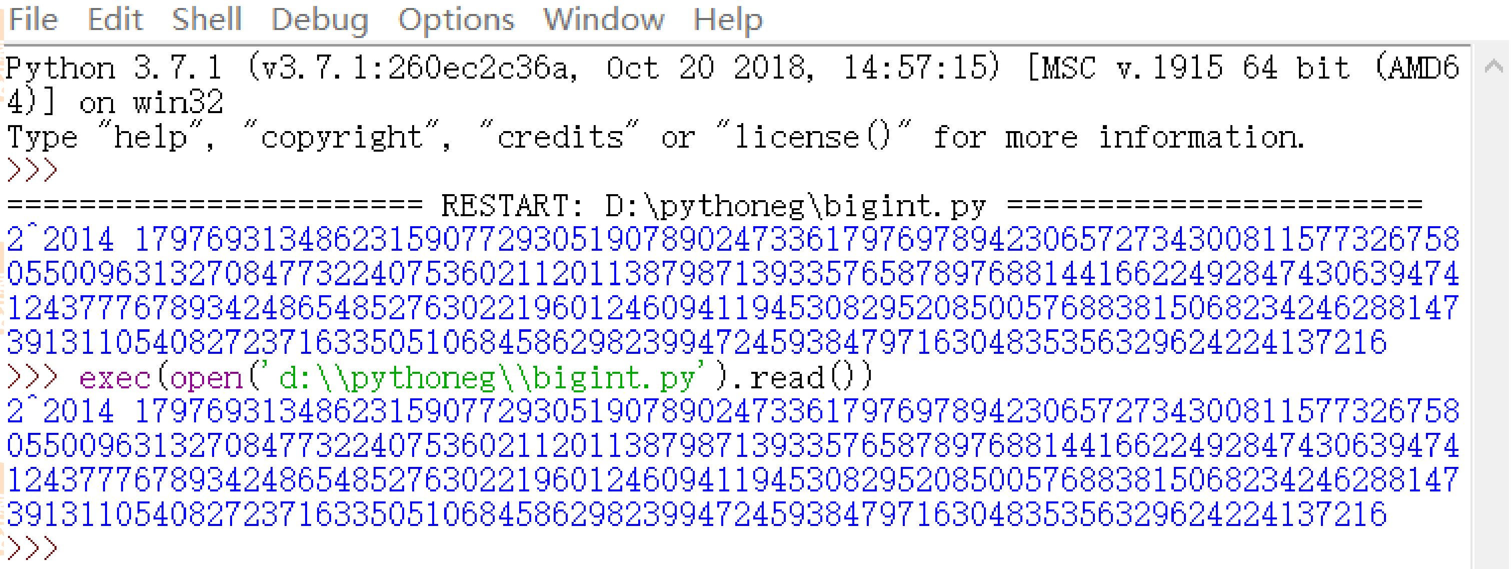
- **Close** IDLE
  - quit()
  - Close the windows

Edit and run .py file in IDLE

- **Write .py program in IDLE and Run**



```
print('2^2014', 2**1024)
```



```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [MSC v.1915 64 bit (AMD6
4)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======================= RESTART: D:\pythoneg\bigint.py =======================
2^2014 179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624224137216
>>> exec(open('d:\\pythoneg\\bigint.py').read())
2^2014 179769313486231590772930519078902473361797697894230657273430081157732675805500963132708477322407536021120113879871393357658789768814416622492847430639474124377767893424865485276302219601246094119453082952085005768838150682342462881473913110540827237163350510684586298239947245938479716304835356329624224137216
>>>
```

4.2 Python Basic

The `print` Statement

`print(”`**`text`**`")`
`print()` (a blank line)

- Escape sequences such as `\"` are the same as in Java
- Strings can also start/end with `'`

swallows.py

```
print("Hello, world! ")
print()
print("Suppose two swallows \"carry\" it together.")
print('African or "European" swallows?')
```

Comments

# comment text (one line)
# must start each line of comments with the pound sign

swallows2.py

```
# Suzy Student, CSE 142, Fall 2097
# This program prints important messages.
print("Hello, world!")
Print()                  # blank line
print("Suppose two swallows \"carry\" it together.")
Print('African or "European" swallows?')
```

Functions

- **Function**: Equivalent to a static method in Java.
- Syntax:

```
def name():
    statement
    statement
    ...
    statement
```

hello2.py

```python
# Prints a helpful message.
def hello():
    print("Hello, world!")

# main (calls hello twice)
hello()
hello()
```

  - Must be declared above the 'main' code
  - Statements inside the function must be indented

Whitespace Significance

- Python uses indentation to indicate blocks, instead of `{}`
  - Makes the code simpler and more readable
  - In Java, indenting is optional. In Python, you **must** indent.
  - You may use either tabs or spaces, but you **must** be consistent

hello3.py

```
# Prints a welcoming message.
def hello():
    print("Hello, world!")
    print("How are you?")

# main (calls hello twice)
hello()
hello()
```

Tabs

shell

```
# Prints a helpful message.
>>> def indent_reminder():
... print("Remember, you must indent!")
  File "<stdin>", line 2
    print("Remember, you must indent!")
        ^
IndentationError: expected an indented block
>>> def indent_reminder():
...     print("Remember, you must indent!")
...
>>> indent_reminder()
Remember, you must indent!
>>>
```

Tabs or spaces

shell

```
# Prints a helpful message.
>>> def indententation_errors():
...     print("this was indented using a tab")
...     print("this was indented using four spaces")
  File "<stdin>", line 3
    print("this was indented using four spaces")
                                               ^
IndentationError: unindent does not match any outer
indentation level
>>> def indententation_errors():
...     print("this was indented using a tab")
...     print("so this must also use a tab")
...
>>> def more_indentation_tricks():
...     print("If I use spaces to indent here.")
...     print("then I must use spaces to indent here.”)
>>>
```

Identifiers

- *Identifiers* are names of various program elements in the code that uniquely identify the elements. They are the names of things like variables or functions to be performed. They're specified by the programmer and should have names that indicate their purpose.

- In Python, identifiers are case sensitive
  - Are made of letters, digits and underscores
  - Must begin with a letter or an underscore
  - Examples: temperature, myPayrate, score2

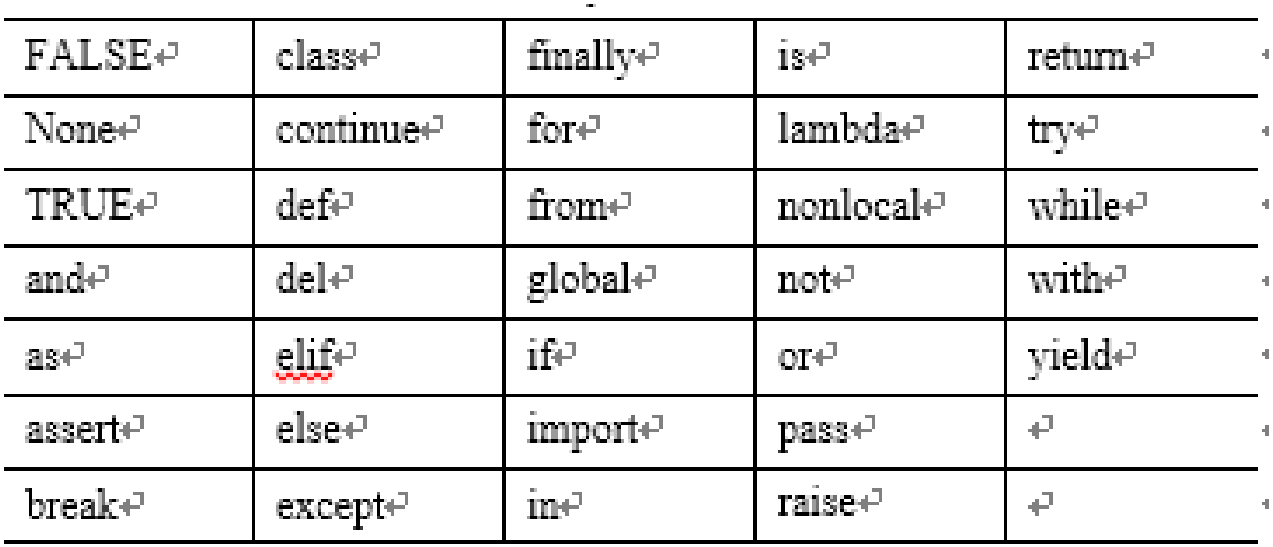
Which identify is correct?

- a_int
- a_float
- Str1
- _strname
- Func1
- 99var
- It’sOK
- for

Keywords

- *Keywords* are reserved words that have special meaning in the Python language. Because they are reserved, they can not be used as identifiers. Examples of keywords are *if, while, class, import*.

Keywords in Python



FALSE	class	finally	is	return
None	continue	for	lambda	try
TRUE	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Python naming convention

- Package and Module Names
  - short, all-lowercase names. Underscores can be used in the module name if it improves readability.
- Class Names
  - Class names should normally use the CapWords convention.
- Method Names and Instance Variables
  - use lowercase with words separated by underscores

Variables in Python

- A variable has
  - A name – identifier
  - A data type - int, float, str, etc.
  - Storage space sufficient for the type.

Numeric Data Types

- int

  This type is for whole numbers, positive or negative. Examples: 23, -1756

- float

  This type is for numbers with possible fraction parts. Examples: 23.0, -14.561

Integer operators

The operations for integers are:

+ for addition
- for subtraction
* for multiplication
/ for integer division: The result of 14/5 is 2
% for remainder: The result of 14 % 5 is 4

- *, /, % take precedence over +, -
  x + y * z will do y*z first

- Use parentheses to dictate order you want.
  (x+y) * z will do x+y first.

Integer Expressions

- Integer expressions are formed using
  - Integer Constants
  - Integer Variables
  - Integer Operators
  - Parentheses

Python Assignment Statements

- In Python, = is called the *assignment operator* and an *assignment statement* has the form

  <variable> = <expression>

- Here
  - <variable> would be replaced by an actual variable
  - <expression> would be replaced by an expression
- Python: age = 19

Python Assignment Statement

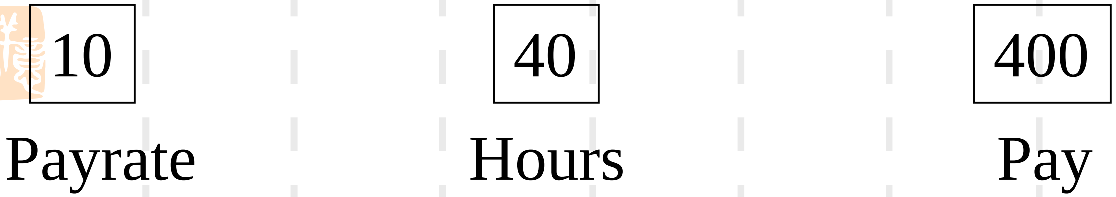
- Syntax: <variable> = <expression>
  - Note that variable is on left

- Semantics:

  Compute value of expression

  Store this as new value of the variable

- Example: Pay = PayRate * Hours



10

Payrate

40

Hours

400

Pay

What about floats?

- When computing with floats, / will indicate regular division with fractional results.
- Constants will have a decimal point.
- 14.0/5.0 will give 2.8 while 14/5 gives 2.

Comments

- Often we want to put some documentation in our program. These are comments for explanation, but not executed by the computer.

- If we have # anywhere on a line, everything following this on the line is a comment – ignored

Numerical Input

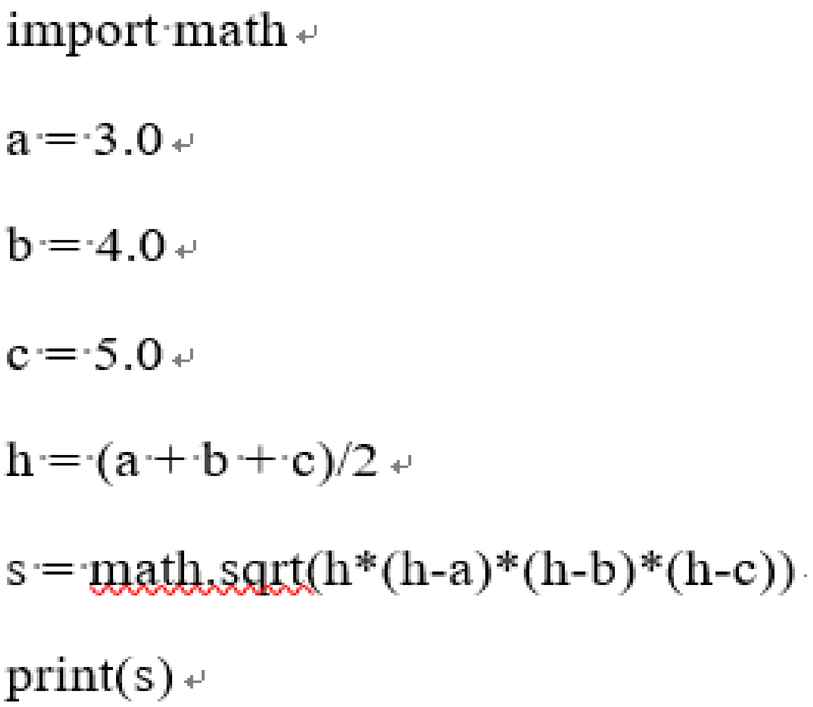
- To get numerical input from the user, we use an assignment statement of the form

```
<variable> = input(<prompt>)
```

- Here
  - <prompt> would be replaced by a prompt for the user inside quotation marks
  - If there is no prompt, the parentheses are still needed
- Semantics
  - The prompt will be displayed
  - User enters number
  - Value entered is stored as the value of the variable

Overview of a Python program

- Given the three edges of a triangle, compute its area of the triangle.（area.py）。
- Hint the area s： s=$\sqrt{h*(h-a)*(h-b)*(h-c)}$
- where h is the half of the circumference



```
import math

a = 3.0

b = 4.0

c = 5.0

h = (a + b + c)/2

s = math.sqrt(h*(h-a)*(h-b)*(h-c))

print(s)
```

Python structure

- modules: Python source files or C extensions
  - import, top-level via from, reload
- statements
  - control flow
  - expression
  - indentation matters – instead of {}
- Expressions
  - create objects
- objects
  - everything is an object
  - automatically reclaimed when no longer needed

It’s all objects…

- Everything in Python is really an object.
  - We’ve seen hints of this already…

    ```
    “hello”.upper()
    list3.append(‘a’)
    dict2.keys()
    ```

  - These look like Java or C++ method calls.
  - New object classes can easily be defined in addition to these built-in data-types.
- In fact, programming in Python is typically done in an object oriented fashion.

Defining a Class

- A *class* is a special data type which defines how to build a certain kind of object.
- The *class* also stores some data items that are shared by all the instances of this class
- *Instances* are objects that are created which follow the definition given inside of the class
- Python doesn’t use separate class interface definitions as in some languages
- You just define the class and then use it

Methods in Classes

- Define a *method* in a *class* by including function definitions within the scope of the class block
- There must be a special first argument **`self`** in *all* of method definitions which gets bound to the calling instance
- There is usually a special method called **`__init__`** in most classes
- We’ll talk about both later…

A simple class def: *student*

```
class student:
  """A class representing a
  student """
  def __init__(self,n,a):
      self.full_name = n
      self.age = a
  def get_age(self):
      return self.age
```

Instantiating Objects

- There is no “new” keyword as in Java.
- Just use the class name with ( ) notation and assign the result to a variable
- `__init__` serves as a constructor for the class. Usually does some initialization work
- The arguments passed to the class name are given to its `__init__()` method
- So, the __init__ method for student is passed “Bob” and 21 and the new class instance is bound to b:

Instantiating Objects

- There is no “new” keyword as in Java.
- Just use the class name with ( ) notation and assign the result to a variable
- `__init__` serves as a constructor for the class. Usually does some initialization work
- The arguments passed to the class name are given to its `__init__()` method
- So, the __init__ method for student is passed “Bob” and 21 and the new class instance is bound to b:

```
b = student(“Bob”, 21)
```

Constructor: __init__

- An `__init__` method can take any number of arguments.
- Like other functions or methods, the arguments can be defined with default values, making them optional to the caller.

- However, the first argument `self` in the definition of __init__ is special…

Self

- The first argument of every method is a reference to the current instance of the class
- By convention, we name this argument ***`self`***
- In `__init__`, *`self`* refers to the object currently being created; so, in other class methods, it refers to the instance whose method was called
- Similar to the keyword *this* in Java or C++
- But Python uses *self* more often than Java uses *this*

Self

- Although you must specify `self` explicitly when *defining* the method, you don't include it when *calling* the method.
- Python passes it for you automatically

Defining a method:

(*this code inside a class definition.)*

```
def set_age(self, num):
  self.age = num
```

Calling a method:

```
>>> x.set_age(23)
```

Deleting instances: No Need to “free”

- When you are done with an object, you don’t have to delete or free it explicitly.
- Python has automatic garbage collection.
- Python will automatically detect when all of the references to a piece of memory have gone out of scope. Automatically frees that memory.
- Generally works well, few memory leaks
- There’s also no “destructor” method for classes

Access to Attributes and Methods

```
class student:
  “““A class representing a student
  ”””
  def __init__(self,n,a):
      self.full_name = n
      self.age = a
  def get_age(self):
      return self.age
```

Traditional Syntax for Access

```
>>> f = student(“Bob Smith”, 23)

>>> f.full_name # Access attribute
“Bob Smith”

>>> f.get_age() # Access a method
23
```

Two Kinds of Attributes

- The non-method data stored by objects are called attributes
- *Data* attributes
  - Variable owned by a *particular instance* of a class
  - Each instance has its own value for it
  - These are the most common kind of attribute
- *Class* attributes
  - Owned by the *class as a whole*
  - *All class instances share the same value for it*
  - Called “static” variables in some languages
  - Good for (1) class-wide constants and (2) building counter of how many instances of the class have been made

Data Attributes

- Data attributes are created and initialized by an `__init__()` method.
  - Simply assigning to a name creates the attribute
  - Inside the class, refer to data attributes using **`self`**
    - for example, **`self.full_name`**

```
class teacher:
  "A class representing teachers."
  def __init__(self,n):
        self.full_name = n
  def print_name(self):
        print self.full_name
```

Class Attributes

- Because all instances of a class share one copy of a class attribute, when *any* instance changes it, the value is changed for *all* instances
- Class attributes are defined *within* a class definition and *outside* of any method
- Since there is one of these attributes *per class* and not one *per instance*, they're accessed via a different notation:
  - Access class attributes using **`self.__class__.name`** notation -- This is just one way to do this & the safest in general.

```
class sample:
   x = 23
   def increment(self):
      self.__class__.x += 1
```

```
>>> a = sample()
>>> a.increment()
>>> a.__class__.x
24
```

Data vs. Class Attributes

```
class counter:
   overall_total = 0
        # class attribute
   def __init__(self):
       self.my_total = 0
         # data attribute
   def increment(self):
      counter.overall_total = \
      counter.overall_total + 1
      self.my_total = \
      self.my_total + 1
```

```
>>> a = counter()
>>> b = counter()
>>> a.increment()
>>> b.increment()
>>> b.increment()
>>> a.my_total
1
>>> a.__class__.overall_total
3
>>> b.my_total
2
>>> b.__class__.overall_total
3
```

Modules

- When a Python program starts it only has access to a basic functions and classes.

  (“int”, “dict”, “len”, “sum”, “range”, ...)
- “Modules” contain additional functionality.
- Use “import” to tell Python to load a module.

```
>>> import math
>>> import nltk
```

import the math module

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.cos(0)
1.0
>>> math.cos(math.pi)
-1.0
>>> dir(math)
['__doc__', '__file__', '__name__', '__package__', 'acos', 'acosh',
'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'exp', 'fabs', 'factorial', 'floor', 'fmod',
'frexp', 'fsum', 'hypot', 'isinf', 'isnan', 'ldexp', 'log', 'log10',
'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc']
>>> help(math)
>>> help(math.cos)
```

Defining Functions

Function definition begins with “def.”

Function name and its arguments.

```
def get_final_answer(filename):
    “““Documentation String”””
    line1
    line2
    return total_counter
```

Colon.

The indentation matters…
First line with less
indentation is considered to be
outside of the function definition.

The keyword ‘return’ indicates the value to be sent back to the caller.

No header file or declaration of types of function or arguments

Python and Types

- **Dynamic typing**: Python determines the data types of *variable bindings* in a program automatically
- **Strong typing:** But Python's not casual about types, it enforces the types of *objects*
- For example, you can't just append an integer to a string, but must first convert it to a string

```
 x = "the answer is "  # x bound to a string
y = 23          # y bound to an integer.
print x + y    # Python will complain!
```

Calling a Function

- The syntax for a function call is:

```
>>> def myfun(x, y):
        return x * y
>>> myfun(3, 4)
12
```

- Parameters in Python are *Call by Assignment*
  - Old values for the variables that are parameter names are hidden, and these variables are simply made to *refer to* the new values
  - All assignment in Python, including binding function parameters, uses *reference semantics.*

Functions without returns

- *All* functions in Python have a return value, even if no *return* line inside the code
- Functions without a *return* return the special value *None*
  - *None* is a special constant in the language
  - *None* is used like *NULL*, *void*, or *nil* in other languages
  - *None* is also logically equivalent to False
  - The interpreter's REPL doesn't print *None*

Default Values for Arguments

- You can provide default values for a function’s arguments
- These arguments are optional when the function is called

```
>>> def myfun(b, c=3,
d=“hello”):
              return b + c
>>> myfun(5,3,”hello”)
>>> myfun(5,3)
>>> myfun(5)
```

Keyword Arguments

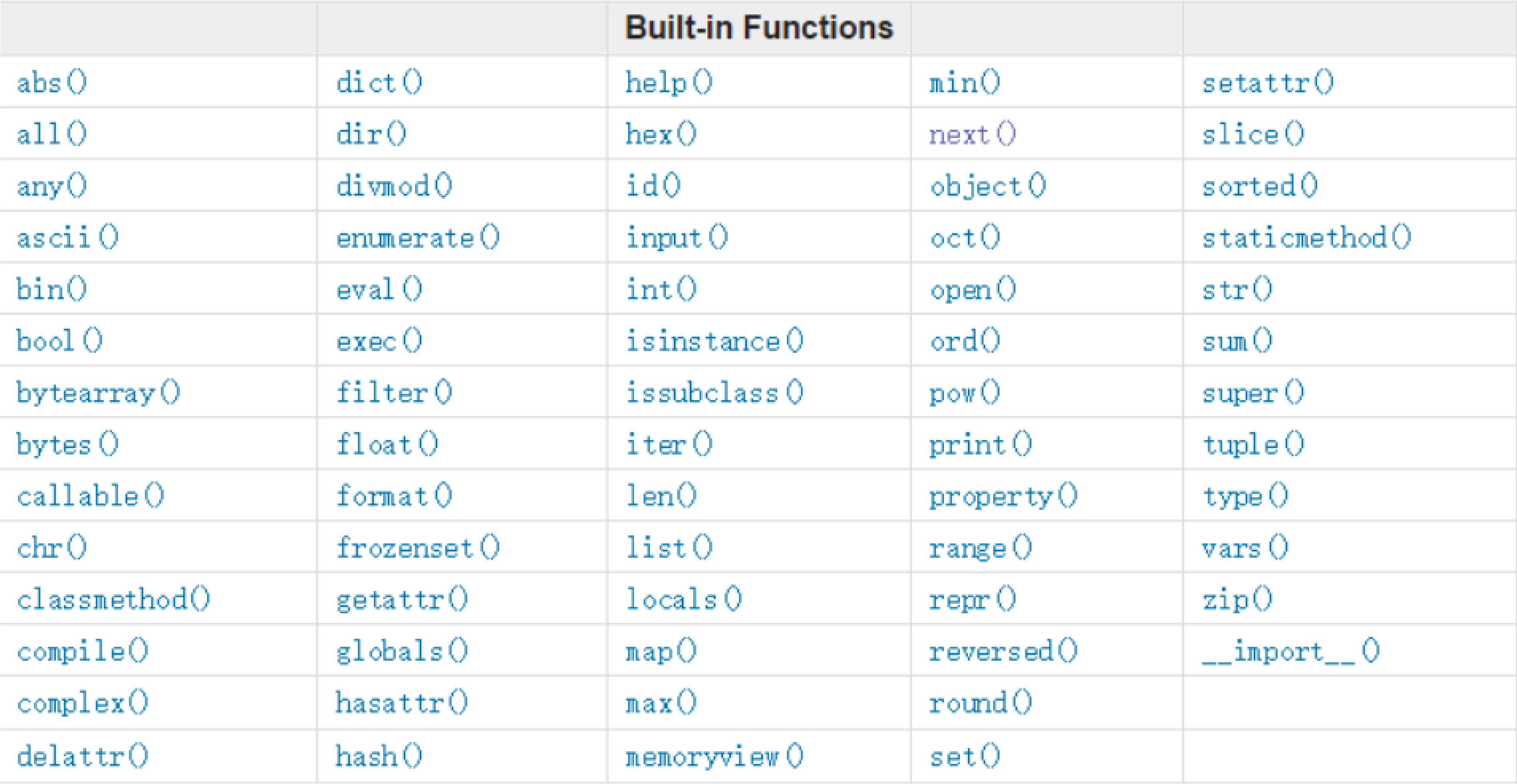
- Can call a function with some/all of its arguments out of order as long as you specify their names

```
>>> def foo(x,y,z): return(2*x,4*y,8*z)
>>> foo(2,3,4)
(4, 12, 32)
>>> foo(z=4, y=2, x=3)
(6, 8, 32)
>>> foo(-2, z=-4, y=-3)
(-4, -12, -32)
```

- Can be combined with defaults, too

```
>>> def foo(x=1,y=2,z=3):
 return(2*x,4*y,8*z)
>>> foo()
(2, 8, 24)
>>> foo(z=100)
(2, 8, 800)
```

Built-in Functions



		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

`id`(*object*)

Return the “identity” of an object. This is an integer which is guaranteed to be unique and constant for this object during its lifetime.

CPython implementation detail: This is the address of the object in memory.

dir(object):tries its best to gather information from the object’s

type(object) With one argument, return the type of an *object*.

Assignment

- You can assign to multiple names at the same time

```
>>> x, y = 2, 3
>>> x

>>> y
3
```

This makes it easy to swap values

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

- Assignments can be chained

```
>>> a = b = x = 2
```

Numeric Data Types

- Whole numbers are represented using the *integer* (*int* for short) data type.
- These values can be positive or negative whole numbers.

Numeric Data Types

- Numbers that can have fractional parts are represented as *floating point* (or *float*) values.
- How can we tell which is which?
  - A numeric literal without a decimal point produces an int value
  - A literal that has a decimal point is represented by a float (even if the fractional part is 0)

Numeric Data Types

- Python has a special function to tell us the data type of any value.

```
>>> type(3)
<class 'int'>
>>> type(3.1)
<class 'float'>
>>> type(3.0)
<class 'float'>
>>> myInt = 32
>>> type(myInt)
<class 'int'>
>>>
```

Numeric Data Types

- Operations on ints produce ints, operations on floats produce floats (except for /).

```
>>> 3.0+4.0
7.0
>>> 3+4
7
>>> 3.0*4.0
12.0
>>> 3*4
12
>>> 10.0/3.0
3.3333333333333335
>>> 10/3
3.3333333333333335
>>> 10 // 3
3
>>> 10.0 // 3.0
3.0
```

Numeric Data Types

- Integer division produces a whole number.
- That's why 10//3 = 3!
- Think of it as 'gozinta', where 10//3 = 3 since 3 gozinta (goes into) 10 3 times (with a remainder of 1)
- 10%3 = 1 is the remainder of the integer division of 10 by 3.
- a = (a/b)(b) + (a%b)

input

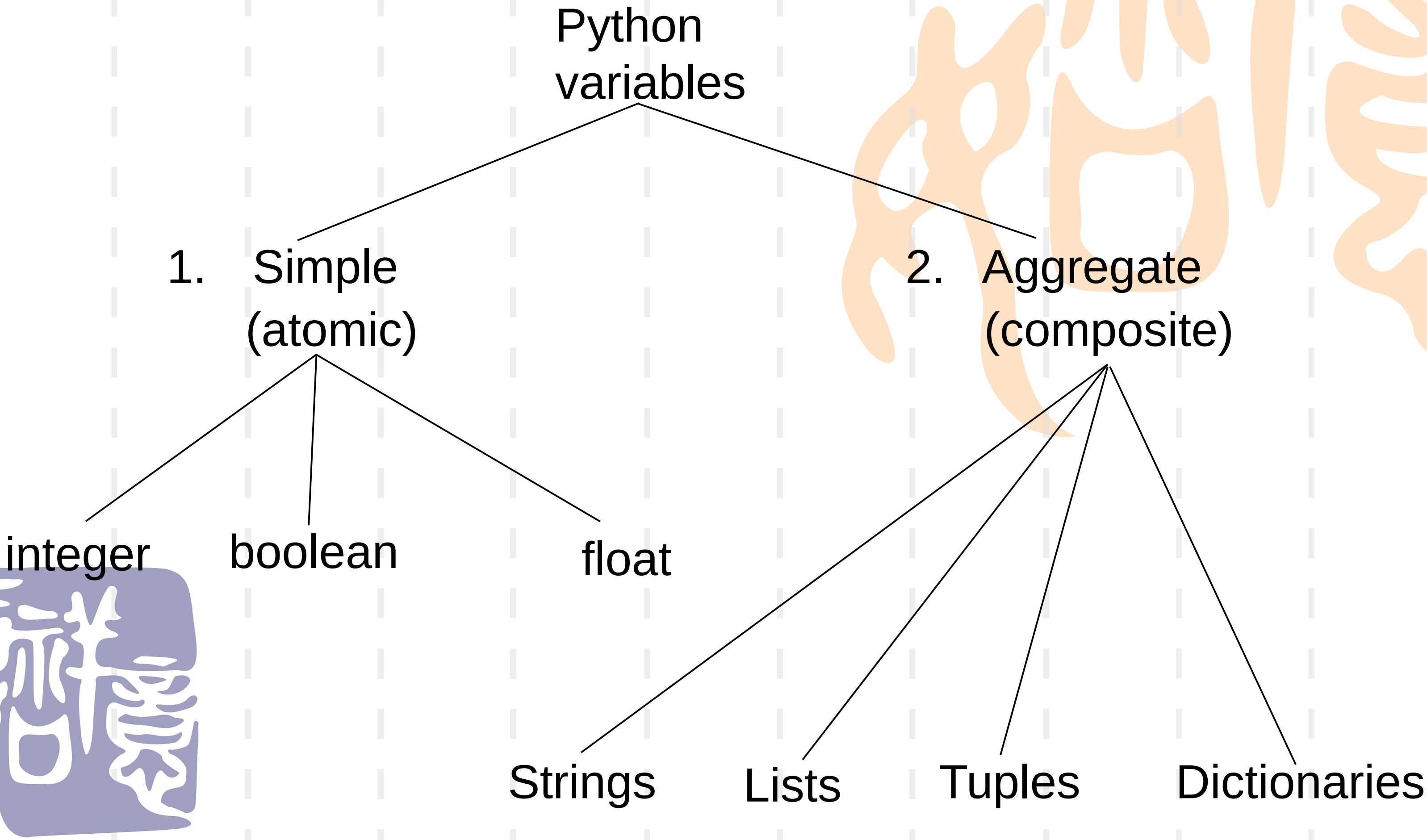
- `input` : Reads a number from user input.
  - You can assign (store) the result of `input` into a variable.
  - Example:

```
age = input("How old are you? ")
print "Your age is", age
print "You have", 65 - age, "years
until retirement"
```

Output:

```
How old are you? 53
Your age is 53
You have 12 years until retirement
```

Types Of Variables



Python variables

1. Simple (atomic)

2. Aggregate (composite)

integer

boolean

float

Strings

Lists

Tuples

Dictionaries