Python

In 8 Hours

For Beginners

Learn Python Fast!

Ray Yao

Python
In 8 Hours

By Ray Yao

For Beginners

Learn Python Fast!

About the Author

Certified PHP engineer by Zend, USA

Certified JAVA programmer by Sun, USA

Certified SCWCD developer by Oracle, USA

Certified A+ professional by CompTIA, USA

Certified ASP. NET expert by Microsoft, USA

Certified MCP professional by Microsoft, USA

Certified TECHNOLOGY specialist by Microsoft, USA

Certified NETWORK+ professional by CompTIA, USA

Preface

This book is a useful book for beginners. You can learn complete primary knowledge of Python fast and easily. The straightforward definitions, the plain and simple examples, the elaborate explanations and the neat and beautiful layout feature this helpful and educative book. You will be impressed by the new distinctive composing style. Reading this book is a great enjoyment! You can master all essential Python skill quickly.

Source Code for Download

This book provides source code for download; you can download the source code for better study, or copy the source code to your favorite editor to test the programs. The download link of the source code is at the last page of this book.

Start coding today!

Table of Contents

Hour 1

Start Python

What is Python?

Python is a general-purpose, object-oriented and open source computer programming language, it is a high-level, human-readable and a corresponding set of software tools and libraries.

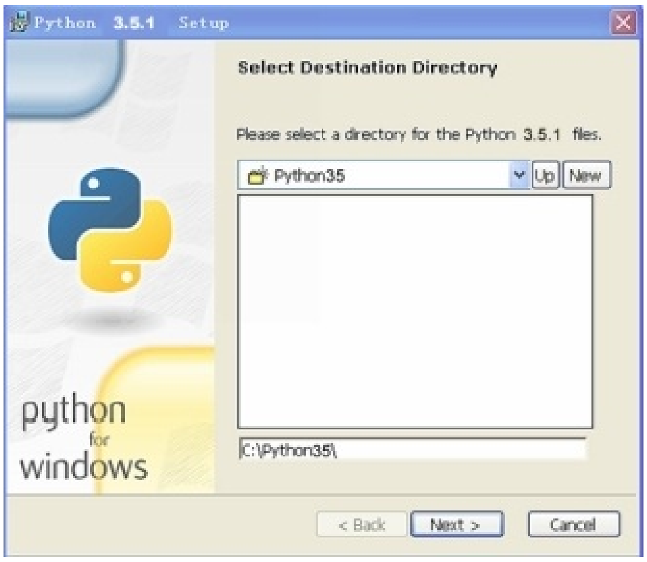
Download Python

Free download Python installer at:

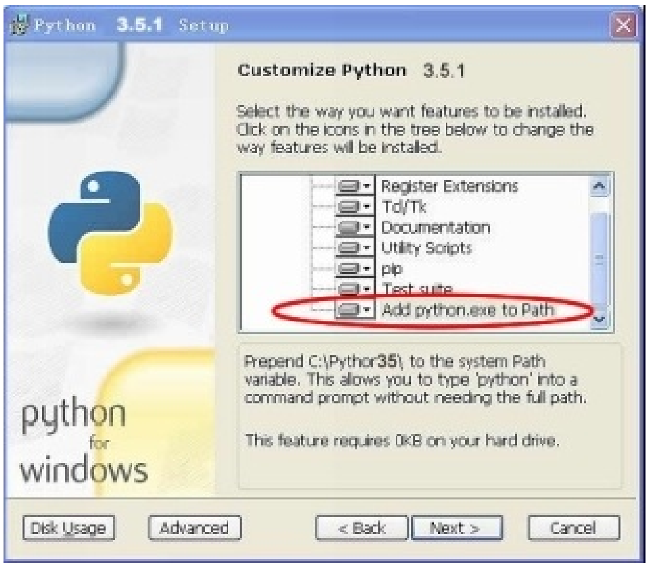
https://www.python.org/downloads/

Install Python

When the download finishes, run the installer by double-clicking it. Install Python to your local computer, For example: at **C:\Python35**. Click "Next".


Python 3.5.1 Setup
Select Destination Directory
Please select a directory for the Python 3.5.1 files.
Python35
Up
New
C:\Python35\
< Back
Next >
Cancel
python for windows

Make sure to select the option "Add Python**.**exe to Path".



Select the way you want features to be installed. Click on the icons in the tree below to change the way features will be installed.

Prepend C:\Python35\ to the system Path variable. This allows you to type 'python' into a command prompt without needing the full path.

This feature requires 0KB on your hard drive.

Please complete the installation of Python.

Run Python

Please click “Start > Programs > Python3.5 > IDLE (Python GUI)”.

Or click “Start > Programs > Python3.5 > Python (Command Line)”.

You will see Python prompt:

```
>>>
```

First Python Program

```
print( )
```

“print ( )” is used to print text or string to the screen.

Example 1.1

```
>>> print ("Hello World!")
```

Output: Hello World!

Example 1.2

```
>>> print (“Python is a very good language!”)
```

Output: Python is a very good language!

```
>>> print (“Learn Python in 8 Hours!”)
```

Output: Learn Python in 8 hours.

Explanation:

“>>>” is a Python primary prompt.

“print ( )” displays the text.

The Shell Prompt

```
>>>
```

“>>>” is the Python interactive command shell prompt, requests the input from user.

For example:

```
>>> 10 + 8
18
>>> 100 - 2
98
>>> ” Hello ” + ” World! “
‘ Hello  World! ‘
>>> ” Very Good! ” * 3
‘ Very Good!  Very Good!  Very Good! ‘
```

Explanation:

“>>>” prompts that you can input data, and press Enter.

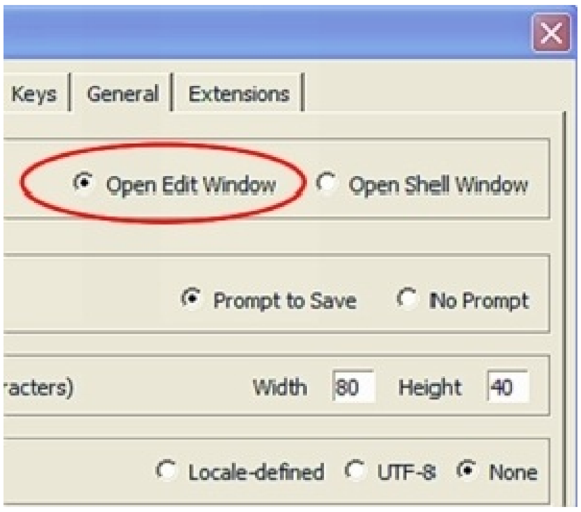
The lines without >>> are responded by Python.

Configure Editor

In order to run the whole Python program instead line by line, or easily copy or paste whole code, you need to configure python editor.

Please click "Start > All Programs > Python3.5 > IDLE (Python GUI)".

Click "Option > Configure IDLE > General > Open Edit Window" > OK.



After selecting "Open Edit Window", you can run the whole Python program instead line by line. Press "**F5**" key to run the program.

(Run>Run Module)

Note:

Option "Open Edit Window" runs the program by whole code.

Execute the whole program by pressing **F5** key.

(Run>Run Module)

Option “Open Shell Window” runs the program line by line.

Execute the one line code by pressing **Enter** key.

Variables

A “variable” is a container that stores a data value. The variable value may change when program is running.

```
variableName = value

variableName1 = variableName2 = value

variableName1,  variableName2 = value1,
value2
```

“variableName” is a variable name.

Example 1.3

```
var = 100

var1 = var2 = var3 = 100

var1, var2, var3 = 100, 200, 300
```

Explanation:

“var = 100” defines a variable named “var”, whose value is 100.

“var1 = var2 = var3 = 100” assigns the value “100” to “var1”, var2”, “var3”.

“var1, var2, var3 = 100, 200, 300” respectively assigns the value “100, 200,300” to “var1”, var2”, “var3”.

Variables & Comment

A variables value can be used in Python program.

```
variableName = value

print (variableName)
```

# Symbol can be used in a comment.

```
# comment
```

Example 1.4

```
var01 = 100
print (var01)       # Output: 100
```

Example 1.5

```
var02 = “Python is very good!”
print (var02)      # Output: Python is very good!
```

Explanation:

“print ( )” is used to display contents.

# is a symbol of comment.

“# Output: 100” is a comment.

“# Output: Python is very good!” is a comment.

Arithmetic Operator

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

Note:

“%” modulus operator divides the first operand by the second operand, returns the remainder.

“//” works like “/”, but returns an integer.

“**” returns the result of the first operand raised to the power of the second operand.

Example 1.6

```
a = 100 + 200

b = 72/9

c = 25 % 7

d = 7 // 3

e = 8 ** 2

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

Output:

```
(300, 8, 4, 2, 64)
```

Explanation:

“100 + 200” returns 300

“72/9” returns 8

“25 **%** 7” returns 4.

“7 // 3” returns 2.

“8 ** 2” returns 64.

Assignment Operators

“x += y” equals “x = x + y”, please see the following chart:

Operators	Examples:	Equivalent:
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y
//=	x//=y	x=x//y
**=	x**=y	x=x**y

Example 1.7

```
x=20

y=2

x %= y

print x
```

Output:

```
0
```

Explanation:

“x %= y” equals “x = x % y”

Example 1.8

```
m=100

n=18

m%=n

print m
```

Output:

```
10
```

Explanation:

“m %= n” equals “m = m % n”.

Comparison Operators

Operators	Running
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
==	equal
!=	not equal

After using comparison operators, the result will be true or false.

Example 1.9

```
a=100

b=200

result1= (a>b)

print(result1)

result2= (a==b)

print(result2)

result3= (a!=b)

print(result3)
```

Output:

```
False

False

True
```

Explanation:

result = (a>b)   # test 100>200   outputs false.

result = (a==b) # test 100==200 outputs false.

result = (a**!**=b) # test 100**!**=200   outputs true.

Logical Operators

Operators	Equivalent
and	logical AND
or	logical OR
not	logical NOT

After using logical operators, the result will be True or False.

Example 1.10

```
x=True

y=False

a=x and y

print (a)


b=x or y

print (b)


c=not x

print (c)
```

Output:

```
False

True

False
```

Explanation:

True **and** True returns true	True **and** False returns False	False **and** False returns False
True **or** True returns True	True **or** False returns True	False **or** False return False
not False returns True	**not** True returns False	

Conditional Operator

The syntax of conditional operator looks like this:

(if-true-do-this) **if** (test-expression) **else** (if-false-do-this)

(test-expression) looks like a<b, x!=y, m==n. etc.

Note: The syntax of conditional operator in Python is different from the " **? :** " ternary operator in C++ or Java.

Example 1.11

```
a=100

b=200

result = “apple” if (a<b) else “banana”

print(result)
```

Output: apple

Explanation:

The conditional operator use (a<b) to test the “a” and “b”, because “a” is less than “b”, it is true. Therefore, the output is “apple”.

Convert Data Type

Function	Operation
int(x)	convert x to an integer number
str(x)	convert x to a string
chr(x)	convert x to a character
float(x)	convert x to a floating point number
hex(x)	convert x to a hexadecimal string
oct(x)	convert x to a an octal string
round(x)	round a floating-point number x.
type(x)	detect x data type

Example 1.12

```
num1 = int(8.67)

print num1     # returns 8

num2 = round(8.67)

print num2    # returns 9.0

num3 = float(5)

print num3    # returns 5.0
```

Explanation:

dataType( ) can convert the data type of a value.

Exercises: Ticket Fare

If…else…

Please click “Start > Programs > Python3.5 > IDLE (Python GUI)”.

Write the following code to the IDLE editor:

```
age = 15
ticket = “Child Fare” if (age < 16 ) else “Adult Fare”
print(ticket)
```

Save the file, and run the program by pressing **F5** key.

(Run>Run Module).

Output:

```
Child Fare
```

Hour 2
Statement

If Statement

```
if  test-expression:
statements
```

“if statement” executes statement only if a specified condition is true, does not execute any statement if the condition is false.

Example 2.1

```
a = 200

b = 100

if a > b:

   print (“a is greater than b.”)
```

Output:

```
a is greater than b.
```

Explanation

a>b is a test expression, namely tests 200>100, if it returns true, it will execute the code “print( )”, if it returns false, it will not execute the code “print( )”.

If-else Statement

```
if  test-expression:
  statements     # run when text-expression is true
else:
  statements     # run when text-expression is false
```

Example 2.2

```
a = 100

b = 200

if a > b:

    print (“a is greater than b.”)

else:

    print (“a is less than b”)
```

Output:

```
a is less than b.
```

Explanation:

a>b is a test expression, namely tests 100>200, if it returns true, it will execute the code “print(‘a is greater than b’)”, if it returns false, it will execute the code “print(‘a is less than b’)”

Indentation

In Python, indentation is used to mark a block of code. In order to indicate a block of code, you should indent each line of the block of code by **four** spaces, which is a typical amount of indentation in Python.

For Example:

```
a = 100

b = 200

if a > b:

    print (“a is greater than b.”)      # indent four spaces

else:

    print (“a is less than b”)       # indent four spaces
```

The “print()” are indented four spaces. Correct!

Note:

```
if a > b:
print (“a is greater than b.”)      # error!
else:
print (“a is less than b”)      # error!
```

The “print()” are not indented, so errors occur!

If-elif-Statement

```
if   test-expression:
  statements    # run when this text-expression is true
elif   test-expression:
  statements    # run when this text-expression is true
else:
  statements    # run when all text-expressions are false
```

Example 2.3

```
num=200
if num < 100:
    print (“num is less than 100”)
elif 100 < num < 150:
    print (“num is between 100 and 150”)
else:
    print (“num is greater than 150”)
```

Output: num is greater than 150

Explanation:

elif is short for “else if”.

For-In Loops

The for-in loop repeats a given block of codes by specified number of times.

```
for <variable> in <sequence> :
<statements>
```

“variable” stores all value of each item.

“sequence” may be a string, a collection or a range( ) which implies the loop’s number of times.

Example 2.4

```
for str in ‘Good’:
    print ‘Current Character :’, str
```

Output:

```
Current Character: G

Current Character: o

Current Character: o

Current Character: d
```

Explanation:

“**for** str **in** ‘Good’” loops four times because “Good” has 4 characters, “str” stores the value of each characters.

for variable in range( )

for variable in range( ) can generate a sequence number

```
for var in range( n)
for var in range(n1, n2)
```

range(n) generates a sequence from 0 to n-1.

range(n1, n2) generates a sequence from n1 to n2-1.

Example 2.5

```
for var in range(6):
    print var
```

Output: 0,1,2,3,4,5.

```
for num in range(3,10) :
    print num
```

Output: 3,4,5,6,7,8,9.

Explanation:

for variable in range( ) can generate a sequence number

While Loops

While –Loops is used to repeatedly execute blocks of code.

```
while <test-expression> :
<statement>
```

<test-expression> looks like a<100, b!=200. c==d, etc.

Example 2.6

```
n = 0
while n <  9:
    print (n)
    n = n + 1
```

Output:

```
012345678
```

Explanation:

“n < 9” tests the “n” value, if “n” is less than 9, “while-loops” will execute the “print (n)”, and continue to run next loop. Until “n” is greater than or equals 9, “while-loops” will terminate the loop.

“n = n + 1” adds 1 to n in each loop.

Continue

```
continue
```

“continue” can skip the next command and continue the next iteration of the loop.

Example 2.7

```
num=0

while num<10:

num = num + 1

if num==5:

    continue

print num
```

Output: 1234678910

Explanation:

Note that the output has no 5.

“if num==5: continue;” skips the next command “print num” when num is 5, and then continue the next while loop.

Break Statement

```
break
```

“break” keyword is used to stop the running of a loop according to the condition.

Example 2.8

```
num=0

while num<10:

if num==5:

    break

num=num+1

print num
```

Output: 5

Explanation:

“if num==5: break” will run the “break” command if num is 5, the break statement will exit from the while loop, then run “print num ”.

Input Texts (1)

Sometimes users need to input some text by keyboard.

```
variable = input(“prompt”)
```

Note: please use double quote marks to enclose your input.

Example 2.9

```
name = input(“Please input your name: “)

print(“Your name is: ” + name )

age = input(“Please input your age: “)

print(“Your age is: ” + age )
```

Output:

```
Please input your name:   “Jack”

Jack

Please input your age:   “16”

16
```

Explanation:

input( ) can accept the data from user’s keyboard input.

Note: please use double quote marks to enclose your input.

Input Texts (2)

Sometimes users need to input some text by keyboard.

```
variable = raw_input(“prompt”)
```

Note: don’t need double quote marks to enclose your input.

Example 2.9

```
name = raw_input(“Please input your name: “)

print(“Your name is: ” + name )

age = raw_input(“Please input your age: “)

print(“Your age is: ” + age )
```

Output:

```
Please input your name:   Jack

Jack

Please input your age:   16

16
```

Explanation:

raw_input( ) can accept the data from user’s keyboard input.

Note: don’t use double quote marks to enclose your input.

Exercises: Traffic Light

If-elif-else statement

Please click “Start > Programs > Python3.5 > IDLE (Python GUI)”.

Write the following code to IDLE editor:

```
trafficLight = raw_input(“Please input traffic light — red, green or yellow: “)
if trafficLight == “red”:
    print (“The traffic light is ” + trafficLight)
elif trafficLight == “green”:
    print (“The traffic light is ” + trafficLight)
else:
    print (“The traffic light is ” + trafficLight)
```

Save the file, and run the program by pressing **F5** key.

(Run>Run Module).

Output:

```
The traffic light is green.
```

Hour 3

Function

Math Function

Python comes with a lot of various modules of function; one of the most useful modules is Math module. The following table lists the most frequently used math functions.

name	**Description**
abs(n)	absolute value of n
round(n)	round off a floating number n
ceil(n)	ceiling of n
floor(n)	flooring of n
max(n, m)	largest of n and m
min(n, m)	smallest of n and m
degrees(n)	convert n from radians to degrees
log(n)	base e logarithm of n
log(n, m)	base m logarithm of n
pow(n, m)	n to the power of m
sqrt(n)	square root of n
sin(n)	sine of n
cos(n)	cosine of n
tan(n)	tangent of x

Note: you may **import math** before using math function

Example 3.1

```
import math
print “degrees(100) : “, math.degrees(100)
print “degrees(0) : “, math.degrees(0)
print “degrees(math.pi) : “, math.degrees(math.pi)
```

Output:

```
degrees(100) :  5729.57795131

degrees(0) :  0.0

degrees(math.pi) :  180.0
```

Explanation:

degrees(n) convert n from radians to degrees

ceil( ) & floor( )

```
math.ceil( );

math.floor( );
```

“math.ceil( );” returns an integer that is greater than or equal to its argument.

“math.floor( );” returns an integer that is less than or equal to its argument.

Example 3.2

```
import math

print “ceil(9.5) : “, math.ceil(9.5)

print “floor(9.5) : “, math.floor(9.5)
```

Output:

```
ceil(9.5) :  10.0

floor(9.5) :  9.0
```

Explanation:

“math.ceil(num));” returns an integer that is greater than or equal 9.5, the result is 10.0.

“math.floor(num);” returns an integer that is less than or equal 9.5, the result is 9.0.

pow ( ) & sqrt ( )

```
math.pow( );

math.sqrt( );
```

“math.pow ( );” returns the first argument raised to the power of the second argument.

“math.sqrt ( );” returns the square root of the argument.

Example 3.3

```
import math

print “pow(4,2) : “, math.pow(4,2)

print “sqrt(4) : “, math.sqrt(4)
```

Output:

```
pow(4,2) :  16.0

sqrt(4) :  2.0
```

Explanation:

“maht.pow(4,2)” returns the first argument “4” raised to the power of the second argument “2”, the result is 16.0

“math.sqrt(4)” returns the square root of the argument “4”, the result is 2.0

Max ( ) & Min ( )

```
math.max( );

math.min( );
```

“math.max( )” returns the greater one between two numbers.

“math.min( )” returns the less number between two numbers.

Example 3.4

```
print “max(4,2) : “, max(4,2)

print “min(4,2) : “, min(4,2)
```

Output:

```
max(4,2):  4

min(4,2):  2
```

Explanation:

“math.max(x, y)” returns the greater number between 100 and 200, the result is 200.

“math.min(x, y)” returns the less number between 100 and 200, the result is 100.

abs( ) & round( )

```
abs( )

round( )
```

abs( ) returns an absolute value of a number.

round( ) rounds of a floating number

Example 3.5

```
print “abs(-100): “, abs(-100)

print “round(0.555,2): “, round(0.555,2)
```

Output:

```
abs(-100):  100

round(0.555,2):  0.56
```

Explanation:

abs(-100) returns an absolute value 100 without negative sign.

round(0.555, 2) rounds the floating number to 2 places.

Custom Function

(1) A custom function can be created by using “def”

```
def functionName( ):      # define a function
  function body
```

(2) The syntax of calling a function

```
functionName( )       # call a function
```

Example 3.6

```
def myFunction( ):
    print(“This is a custom function.”)


myFunction( )
```

Output:

```
This is a custom function
```

Explanation:

def myFunction( ): defines a function

myFunction( ) calls a function

Function with Arguments

(1) define a function with arguments

```
def functionName(arguments):      # define a function
  function body
```

(2) call a function with arguments

```
functionName(arg)     # call a function
```

Example 3.7

```
def userName(name):
    print( “My name is ” + name)


userName(“Andy”)
```

Output:

```
My name is Andy
```

Explanation:

“def userName(name):” defines an argument “name”.

“userName(“Andy”)” passes “Andy” to argument “name”.

Global & Local Variable

Global Variable is defined outside the function and can be referenced from inside the function.

Local Variable is defined inside the function and cannot be referenced from outside the function.

Example 3.8

```
globalVar = “GV”      # defines a global variable
def testFunction( ):
    print (“The global variable is: ” + globalVar)
    localVar = “LV”      # defines a local variable
    print (“The local variable is: ” + localVar)
testFunction( )    # call a function
```

Output:

```
The global variable is: GV        The local variable is: LV
```

Explanation:

“GV”   is a global variable. “LV” is a local variable.

Global Variable inside Function

If want a local variable to be referenced in everywhere including inside and outside the function, you should use “**global**” keyword before the local variable name.

Example 3.9

```
def tryFunction( ):

   global tryVar     # defines a global inside the function

   tryVar = “This variable can be referenced in everywhere.”

tryFunction( )    # call a function

print (“tryVar: ” + tryVar )     # reference tryVar
```

Output:

```
tryVar: This variable can be referenced in everywhere.
```

Explanation:

“global tryVar **“** defines a global inside the function

“tryVar” can be referenced in everywhere.

Return

“return” specifies a value to be returned to the caller.

Example 3.10

```
def multiply ( n, m ) :
    return n*m
print multiply( 2,100 )
```

Output:

```
200
```

Explanation:

“return n*m” returns its result value to multiply(2,100).

It is equivalent to ***multiply(2, 100) = return n*m***

Main Function

The main( ) function is a default start point of the whole program.

```
def main ():
  function body
```

Note: In Python, main( ) function is optional.

Example 3.11

```
def main():
    pwd = raw_input (“Please enter your password: “)
    if pwd == “12345”:
    print(“Password is correct!”)
    else:
    print(“Password is incorrect!”)
main()
```

Output:

```
Password is correct! / incorrect!
```

Explanation:

main( ) is a default start point by convention.

List all functions in module

```
dir(module)
```

“dir(module)” can list all functions in the specified module.

Example 3.12

```
import math

print dir(math)

.
```

Output:

```
['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2',
'atanh', 'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs',
'factorial', 'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'hypot', 'isinf', 'isnan', 'ldexp',
'lgamma', 'log', 'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'trunc']
```

Explanation:

“print dir(math)” list all functions in math module.

Exercises: Circle Area

Calling function & return

Please click “Start > Programs > Python3.5 > IDLE (Python GUI)”.

Write the following code to IDLE editor:

```
import math
r = input(“Please enter a radius: “)
def circleArea():
    return math.pi*pow(r, 2)  # calculate circle area
print “The circle area is: “, circleArea()
```

Save the file, and run the program by pressing **F5** key.

(Run>Run Module).

Output:

```
Please enter a radius: 3

The circle area is:  28.2743338823
```

Hour 4

Data Structures

List

A list in Python is like an array in Java, is a collection of a series of data. You can add remove or modify elements in List.

```
listName = [val1, val2, val3]
```

Example 4.1

```
month = [“Mon”, “Tue”, “Wed”, “Thu”]     # create a list

print month[0], month[1], month[2], month[3]
```

Output:

```
Mon Tue Wed Thu
```

Explanation:

List “month” has four elements.

The key of Mon is 0.

The key of Tue is 1.

The key of Wed is 2.

The key of Thu is 3.

Note: key start with 0 in List.

List Functions

Function	**Operation**
list.append(n)	Append n to the end of list
list.count(n)	Count how many n
list.index(n)	Return the index of n
list.insert(i,n)	Insert n before index i
list.pop(i)	Remove & return the item at index i
list.remove(n)	Remove the n
list.reverse()	Reverse the sequence of list
list.sort()	Sort the element of list increasingly
list extend(lst)	Append each item of lst to list

Example 4.2

```
list = [0, 1, 2]     # create a list
list.append(3)   # append 3 to the end of list
list.reverse()     # reverse the order of list
print list     # Output: [3, 2, 1, 0]
```

Explanation:

Above code uses List functions.

Know More List

```
list1 + list2    # concatenate two lists

len(list)     # return length of list
```

Example 4.3

```
lst1 = [0, 1, 2]

lst2 = [3, 4, 5]

myList = lst1 + lst2

print “myList: “, myList

print “myList[5]: “,myList[5]

print “len(myList): “, len(myList)
```

Output:

```
myList:  [0, 1, 2, 3, 4, 5]

myList[5]:  5

len(myList):  6
```

Explanation: “list1 + list2” concatenates two lists. “len(list)” returns the length of list”.

Tuple

Tuple ‘s value is unchangeable, it is an immutable List.

```
tupleName = (val1, val2, val3)
```

Example 4.4

```
tpl = (“Mon”, “Tue”, “Wed”, “Thu”)  # create a tuple

print len(tpl)

print tpl.index(“Wed”)
```

Output: 4   2

Explanation:

len(tpl) returns the length of “tpl”.

index(“Wed”) returns the index of the “Wed”.

Example 4.5

tpl = (“Mon”, “Tue”, “Wed”, “Thu”) # create a tuple

print tpl.append(“Fri”)

Output: Error!

Explanation:

Tuple’s value is unchangeable.

Tuple Functions

Function	**Operation**
x in tpl	return true if x is in the tuple
len(tpl)	return length of tuple
tpl.count(x)	count how many x in tuple
tpl.index(x)	return the index of x

Example 4.6

```
colors = (“red”, “yellow”, “green”)

print colors                    # Output: (‘red’, ‘yellow’, ‘green’)

print “yellow” in colors          # Output: True

print len(colors)                # Output:  3

print colors.index(“green”)    # Output:   2

print colors.count(“red”)       # Output:   1
```

Explanation:

Above samples are the demonstrations of tuple functions.

Set

Set’s value is unique; it is a special list whose value is unique.

```
setName = {“dog”, “cat”, “rat”}
```

Example 4.7

```
animal = { “dog”, “cat”, “rat”, “dog”}   # create a set
```

```
print animal

print “cat” in animal

print len(animal)
```

Output:

```
set([‘rat’, ‘dog’, ‘cat’])
True
3
```

Explanation:

In spite of four elements in the Set, the len(animal) still returns 3, because the value of Set is unique. One of the dogs is omitted.

Set Functions

Function	Operation
set.add(n)	Add x to the set
set.update(a, b, c)	Add a, b, c to the set
set.copy( )	Copy the set
set.remove(n)	Remove the item n
set.pop( )	Remove one random item
set1.intersection(set2)	Return items in both sets
set1.difference(set2)	Return items in set1 not in set2

Example 4.8

```
languages = {“ASP”, “PHP”, “JSP”}

languages.add(“C++”)

print languages    # Output:  set([‘ASP’, ‘PHP’, ‘JSP’, ‘C++’])

languages.remove(“PHP”)

print languages    # Output:  set([‘ASP’, ‘JSP’, ‘C++’])
```

Explanation:

Above samples are the demonstrations of set functions.

Dictionary

Dictionary is a data structure for storing pairs of values with the format **key:value**.

```
dictionaryName = { key1: val1,  key2:val2,
key3:val3 }
```

Example 4.9

```
light = {0:“red”, 1:“yellow”, 2:“green”}   # create a dictionary

print light

print light[1]

light[2] = “white”

print light
```

Output:

```
{0: ‘red’, 1: ‘yellow’, 2: ‘green’}

yellow

{0: ‘red’, 1: ‘yellow’, 2: ‘white’}
```

Explanation:

light[2] = “white” assigns the value “white” to “light[2]”.

Dictionary Functions

Function	**Operation**
d.items( )	return key**:**value pairs of d
d.keys()	return keys of d
d.values()	return values of d
d.get(key)	return the values with specified key
d.pop(key)	remove key and return its value
d.clear()	remove all items of d
d.copy()	copy all items of d
d.setdefault(k,v)	set key**:**value to d
d1.update(d2)	add key**:**value in d1 to d2

Example 4.10

```
light = {0:“red”, 1:“yellow”, 2:“green”}

print light.items()  # Output:  [(0, ‘red’), (1, ‘yellow’), (2, ‘green’)]

print light.keys()   # Output:  [0, 1, 2]

print light.values()    # Output:   [‘red’, ‘yellow’, ‘green’]

print light.get(2)    # Output:   green
```

Explanation:

Above samples are the demonstrations of dictionary functions.

Data Structure Review

Structures	**Descriptions**
List	store multiple changeable values
Tuple	store multiple unchangeable values
Set	store multiple unique values
Dictionary	store multiple key:value pairs

Example 4.11

```
myList = [1,2,2,2,3,4,5,6,6,6]

result = set(myList)

print result
```

Output:

```
set([1, 2, 3, 4, 5, 6])
```

Explanation:

“set(myList)” returns multiple unique values.

Exercises: Four Colors

Dictionary Demo

Please click “Start > Programs > Python3.5 > IDLE (Python GUI)”.

Write the following code to IDLE editor:

```
color ={0:“red”, 1:“yellow”, 2:“green”, 3:“white”}
v = color.values()
for c in v:
    print (c)
```

Save the file, and run the program by pressing **F5** key.

(Run>Run Module).

Output:

```
red

yellow

green

white
```

Hour 5

Strings

Operation Strings

String is consisted of a group of characters; its values can be operated by following operators.

Operator	**Description**
+	concatenate strings together
*	repeat a string
[key]	return a character of the string
[key1**:** key2]	return characters from key1 to key2-1
in	check a character existing in a string
not in	check a character not existing in a string
’’’ ’’’	describe a function, class, method…

Example 5.1

```
myString = “Python “+ “is a good language”
print myString        # Output: Python is a good language
print myString[2]      # Output:  t
print ‘P’ in myString      # Output:  True
print myString[7:25]     # Output:   is a good language
```

Explanation:

Note: string[key1**:** key2] returns characters from key1 to key2-1.

Escape Characters

Characters	**Description**
\	escape backslash
\’	escape single quote
\”	escape double quote
\n	new line
\r	return
\t	tab

Example 5.2

```
print(‘Python said: \”Welcome!\”‘)

print (“Python \t is\t OK!”)
```

Output:

```
Python said: “Welcome!”
Python		is		OK!
```

Explanation:

\” escapes the double quote, **\t** is equivalent to tab.

Testing Functions

Testing Functions return either True or False.

Functions	**Return True if…**
isalpha()	return true if all characters are letters
isdigit()	return true if all characters are digits
isdecimal()	return true if all characters are decimals
isalnum()	return true if all characters are numbers or letters
islower()	return true if all characters are lowercase
isupper()	return true if all characters are uppercase
istitle()	return true if the string is title-case string
isspace()	return true if the string contains only whitespace

Example 5.3

```
s1 = “1124324324”

print s1.isdigit()     # Output:   True

s2 = “Chicago”

print s2.istitle()     # Output:   True
```

Explanation:

isdigit() and istitle() are testing functions.

Searching Functions

Functions	Return
find(c)	return the index of first occurrence, or -1
rfind(c)	same as find(), but find from right to left
index(c)	return the index of first occurrence, or alert error
rindex(c)	same as index(), but find from right to left

Example 5.4

```
s1 = “JavaScript”

print s1.find(“a”)    # Output:  1

s2 = “JavaScript”

print s2.rfind(“a”)    # Output:  3

s3 = “abec”

print s3.index(“e”)    # Output:  2
```

Explanation:

find(), rfind() and index() returns the index of specified character.

Formatting Functions

Functions	**Returned String**
center(w, f)	center string with width w and fill with f
ljust(w,f)	left adjust string with width w and fill with f
rjust(w,f)	right adjust string with width w and fill with f

Example 5.5

```
str = “this is a center example”

print “str.center(35, ‘$’) : “, str.center(35, ‘$’)

print “str.ljust(35, ‘$’) : “, str.ljust(35, ‘$’)

print “str.rjust(35, ‘$’) : “, str.rjust(35, ‘$’)
```

Output:

```
str.center(35, ‘$’) :  $$$$$$this is a center example$$$$$

str.ljust(35, ‘$’) :  this is a center example$$$$$$$$$$$

str.rjust(35, ‘$’) :  $$$$$$$$$$$this is a center example
```

Explanation:

Argument “35” means the length of this string.

If filler is an empty character, then it returns whitespaces.

Stripping Functions

Functions	**Returned String**
strip()	remove leading and trailing spaces
lstrip()	remove leading spaces
rstrip()	remove trailing spaces

Example 5.6

```
str = “    This is a strip sample!    “;

print str.lstrip( )

str = “    This is a strip sample!    “;

print str.rstrip( )
```

Output:

```
This is a strip sample!

This is a strip sample!
```

Explanation:

If using strip(“@”), it will remove leading and trailing @.

If using strip( ), it will remove leading and trailing whitespaces.

Splitting Functions

Functions	Returned String
split(separator)	split a string by a separator. (default whitespace as a separator)
partition(separator)	partition a string by a separator to three parts. (head, separator, tail)

Example 5.7

```
str = “Python is a very good language”
print str.split()    # specify whitespace as separator
email = “xxx@yyyyyy.com”
print email.partition(“.”)   # specify “.” as separator
```

Output:

```
[‘Python’, ‘is’, ‘a’, ‘very’, ‘good’, ‘language’]

(‘xxx@yyyyyy’, ‘.‘, ‘com’)
```

Explanation:

Specified separator only comes from the given string. You cannot specify a separator that does not exist in the string.

“email.partition(“.”)” separates the email to three parts. (head, separator, trail)

String Functions (1)

Functions	**Returned Strings**
replace(old, new)	replace every old with new
count(ch)	count the number of the characters
capitalize()	change the first letter to uppercase

Example 5.8

```
str = “jQuery is a great language!”

print str.replace(“great”,“very good”)

print str.count(“a”)

print str.capitalize()
```

Output:

```
jQuery is a very good language!

4

Jquery is a great language!
```

Explanation:

The above samples are demos of three string functions

String Functions (2)

Functions	**Returned Strings**
separater.join()	join the strings by separator
str.swapcase()	swap the letters case of the string
str.zfill(length)	add zeros to the left of the string with length

Example 5.9

```
strDate = “/”.join([“12”, “31”,“2013”])

print strDate

str = “Python”

print str.swapcase()

print str.zfill(10)
```

Output:

```
12/31/2013

pYTHON

0000Python
```

Explanation:

“”/”**.**join([“12”, “31”,“2013”])” separates the date string by “/”.

“str.swapcase()” changes “Python” to “pYTHON”

“str.zfill(10)” fills “0” to the string with length “10” on the left.

Regular Expressions

Regular Expressions are used to match the string with specified pattern, perform the tasks of search, replacement and splitting…

Operators	**Matches**
^	Matches beginning of line.
$	Matches end of line.
.	Matches any single character.
[…]	Matches any single character in brackets.
[^…]	Matches any single character not in brackets
?	Matches 0 or 1 occurrence
+	Matches 1 or more occurrence
*****	Matches 0 or more occurrences
{ n}	Matches exactly n number of occurrences
{ n, m}	Matches at least n and at most m occurrences
a | b	Matches either a or b.
(re)	Groups regular expressions

Example 5.10

[Jj]Query	Matches “JQuery” or “jQuery”
s[ei]t	Matches “set” or “sit”
[0-9]	Matches any numbers
[a-z]	Matches any lowercase letter
[A-Z]	Matches any uppercase letter
[a-zA-Z0-9]	Matches any numbers and letters
[^0-9]	Matches anything other than a number
lady?	Matches “lad” and “lady”.
m+	Matches “m”, “mm”, “mmm” ,”mmmm”……
w{3}	Matches three “w”. e.g. “www”
n{2,4}	Matches 2, 3, or 4 “n”. e.g. “nn”, “nnn”, “nnnn”

\w	Matcheses word characters.
\W	Matcheses non-word characters.
\s	Matcheses space.
\S	Matcheses non-space.
\d	Matcheses numbers.
\D	Matcheses non-numbers.

Example 5.11

\w	Matches a word character: [a-zA-Z_0-9]
\d	Matches [0-9]

Regular Expression in Python

```
re.compile( regular expression)      # return a pattern object

pattern.match(string)      # match the pattern with string
```

Example 5.12

(Assume that only the phone number format **ddd-ddd-dddd** is acceptable.)

```
import re        # import re module
pattern = re.compile("^(\d{3})-(\d{3})-(\d{4})$")
phoneNumber = raw_input("Enter your phone number:")
valid = pattern.match(phoneNumber)
if valid:
    print ("OK! Valid Phone Number!")
else:
    print("No Good! Invalid Phone Number!")
```

Output:

```
Enter your phone number:  123-123-1234

OK! Valid Phone Number!
```

Explanation:

re.compile(“^(\d{3})-(\d{3})-(\d{4})$”) returns a pattern object.

Exercises: Check Your Input

isalpha() Demo

Please click “Start > Programs > Python3.5 > IDLE (Python GUI)”.

Write the following code to IDLE editor:

```
name = raw_input(“Please enter your last name:  “)

isLetter =  name.isalpha()

if isLetter:

    print (“OK! Valid Last Name!”)

else:

    print (“No Good! Invalid Last Name!”)
```

Save the file, and run the program by pressing **F5** key.

(Run>Run Module).

Output:

(Assume inputting “Swift”)

```
Please enter your last name:  Swift

OK! Valid Last Name!
```

Hour 6

Input & Output

Format String

The string output can be formatted as following chart.

Specifier	Description
d	integer
f	float
s	string
o	octal value
x	hexadecimal value
e	exponential
%	“%formatted value” from %original value

Example 6.1

```
num = 9.12345678

print (“String value is: %s” %num)

print (“Float value is: %.3f” %num)    # three decimal places

print (“Exponential value is: %e” %num)

print (“Octal value is: %o” %num)

print (“Hexadecimal value is: %x” %num)

print (“Integer value is: %d” %num)
```

Output:

```
String value is: 9.12345678

Float value is: 9.123

Exponential value is: 9.123457e+00

Octal value is: 11

Hexadecimal value is: 9

Integer value is: 9
```

Explanation:

In (“……%xxx”   %yyy),

%xxx is a formatted value,

%yyy is an original value.

File Directory

To handle with file directory, you need to import os module.

```
path = os.getcwd()   # return current working
directory

os.listdir(path)     # list anything in current directory

os.chdir(path)     # set path as current directory
```

Example 6.2

```
import os
print os.getcwd()      # return current  working directory
path = os.getcwd()
print os.listdir(path)  # return files and sub directories
```

Output:

```
C:\Python35
[‘DLLs’, ‘Doc’, ‘include’, ‘Lib’, ‘libs’, ‘LICENSE.txt’, ‘myFile.txt’, ‘NEWS.txt’,
‘python.exe’, ‘pythonw.exe’,……]
```

Explanation:

“os.listdir(path)” returns all files and sub directories in current directory.

Open File

The syntax to open a file looks like this:

```
open(filename, “argument”)
```

The arguments are listed as following:

arguments	**actions**
r	open file for reading (default)
w	open file for writing
a	open file for appending
+	open file for reading & writing
b	open file in binary mode
t	open file in text mode

Example 6.3 (Assume there is file “myFile.txt” in the same folder with the following file.)

```
f = open(“myFile.txt”)

print f.name      # Output:  myFile.txt
```

Explanation:

“open(“myFile.txt”)” opens myFile.txt

f**.**name returns the file name.

Read file

```
open(“fileName”, “r”)    # using “r” mode for opening file

f.read( )    # read the contents of the file
```

Example 6.4

(Content of “myFile.txt” is: **“Hallo! This is myFile.txt. Welcome!”**)

```
f = open(“myFile.txt”, “r”)    # use “r” mode

print f.name    # Output:  myFile.txt

print (f.read())

f.close()    # close file
```

Output:

```
myFile.txt

Hello! This is myFile.txt. Welcome!
```

Explanation:

open(“myFile.txt”, “r”) uses “r” mode to open the file for reading.

f.read() returns the contents of the file.

Write File

```
open(“fileName”, “w”)     # using “w” mode for opening file

f.write( “text” )      # write text to the file
```

Example 6.5

```
f = open(“myFile.txt”, “w”)     # use “w” mode
print f.name      # Output:  myFile.txt
f.write(“I want to write something to the file.”)
f.close()     # close file
```

Result:

Please open the file “myFile.txt”, you can see the contents:

“I want to write something to the file.”.

Explanation:

open(“myFile.txt”, “w”) uses “w” mode to open the file for writing.

“f.write(“I want to write something to the file.”)” writes new text to the file, and remove original text.

Append Text to File

```
open(“fileName”, “a”)     # using “a” mode to append
text

f.write( “text” )     # write text to the file
```

Example 6.6

```
f = open(“myFile.txt”, “a”)     # use “a” mode

print f.name      # Output:  myFile.txt

f.write(” This is the appended text.”)

f.close()
```

Result:

Check myFile.txt, you can see the text as following:

I want to write something to the file. This is the appended text.

Explanation:

open(“myFile.txt”, “a”) uses “a” mode to open the file for appending new text to the end of the original text.

“a” mode can keep the original text.

Renew Some Text

```
open(“fileName”, “r+”)    # use “r+” mode for reading & writing

f.seek(index)    # set the pointer to specified index
```

Example 6.7

```
f = open(“myFile.txt”, “r+”)      # use “r+” mode

f.seek(10)

f.write(“renew”)      # renew text
```

Output:

```
I want to renew something to the file. This is the appended text.
```

Explanation:

“open(“myFile.txt”, “r+”)” uses “r+” mode for reading & writing.

“f.seek(10)” sets the pointer to the index 10.

“f.write(“renew”)” changes the original text to “renew”.

In myFile.txt. the word “write” has been changed to “renew”. You can see:” I want to **renew** something to the file. This is the appended text.”

Open Web Page

Python programming can open a web page.

```
import webbrowser    # import webbrowser

webbrowser.open("URL")    # open a specified web
page
```

The above codes are very useful for access to a website.

Example 6.8

```
import webbrowser

url = “http://www.amazon.com”

webbrowser.open(url)

print (“You are visiting “+ url)
```

Output:

```
You are visiting http://www.amazon.com
```

(You can see Amazon home page appearing.)

Explanation:

“webbrowser.open(url)” opens a specified web page.

Exercises: Process a File

Write & Read File

(Please create an empty text file “tryFile.txt” in the same directory with the following Python file)

Please click “Start > Programs > Python3.5 > IDLE (Python GUI)”.

Write the following code to IDLE editor:

```
f = open(“tryFile.txt”, “w”)
f.write(“I am learning Python programming!”)
f.close
f =open(“tryFile.txt”, “r”)
print(f.read())
f.close
```

Save the file, and run the program by pressing **F5** key.

(Run>Run Module).

Output:

```
I am learning Python programming!
```

Hour 7

Module & Exception

Module

Module is a file that contains some various functions. Module is used to support another file.

Example 7.1

(The following file **support.py** is a module file)

```
def a():
    print (“This is function a.”)
def b():
    print (“Hello from function b.”)
def c():
    print (“I am function c”)
def d():
    print (“Here is function d”)
def e():
    print (“Hi! function e”)
```

Save the above file named as “**support.py**”.

Import Module (1)

```
import module

moduleName.function()
```

“import module” imports a module to current file.

“moduleName.function()” calls the functions in the module.

The module should be in the same directory with the working file.

Example 7.2

```
import support

support.a()

support.d()
```

Output:

```
This is function a.

Here is function d.
```

Explanation:

“import support” imports module support.py to current file.

“support.a()” calls the function in module support.py.

“support.d()” calls the function in module support.py.

Import Module (2)

```
from module import *

function()
```

“from module import*” imports a module to current file.

“function()” calls the functions in module.

The module should be in the same directory with the working file.

Example 7.3

```
from support import *
c()
e()
```

Output:

```
I am function c.

Hi! function e.
```

Explanation:

“from support import *” import any functions from support.py

If you want to import some specified functions, you should write the code like this: **from support import c, e**

Note: c() calls function other than support.c().

Built-in Module

Python provides many built-in modules to import, such as math module(for math), cgi module(for script), datetime module(for date and time), re module(for regular expression) ……

Example 7.4

```
import math
from datetime import *
print math.sqrt(100)
d = datetime.today()
print (d)
```

Output:

```
10.0

2016-02-21 23:20:29.818000
```

Explanation:

“import math” imports built-in module math.

“from datetime import *” imports built-in module datetime

Exceptions

When an exception occurs and is not caught by the program itself, Python immediately terminates the program and outputs a “traceback”, showing error message.

Example 7.5

```
100/0
```

Output:

```
Traceback (most recent call last):
  File “C:/Python35/exception.py”, line 1, in <module>
    100/0
ZeroDivisionError: integer division or modulo by zero.
```

Explanation:

“Traceback**……**” is an exception message.

Exception message may vary based on different reasons. Other errors may be “SyntaxError”, “IOError”, “ValueError”……

Catch Exceptions

```
try:
……
except  XxxError as message:
……
```

“try block” contains the code that may cause exception.

“except block” catches the error, and handle the exception.

Example 7.6

```
try:
    int("ten")
except ValueError as message:
    print("Exception occurs!", message)
```

Output:

```
('Exception occurs!', ValueError("invalid literal for int() with base 10: 'ten'",))
```

Explanation: In try block, int("ten") cannot be converted to an integer, an exception occurs. except block catches the error, and handles it by showing a error message.

Finally

```
finally:
```

In “try/except” block, “finally” statement is the code that must be executed. The program must run the “finally statement” at last.

Example 7.7

```
while True:
    try:
    num = int(raw_input(“Please enter your ID:  “))
    except ValueError as message:
    print message
    finally:
    print (“Remind: please input number only.”)
```

Output:

```
Please enter your ID:  007hero

invalid literal for int() with base 10: ‘007hero’

Remind: please input number only.
```

Explanation:

int() converts any input data to digits. If some data type cannot be converted to integer, an exception will occur.

The try block contains the code that easily raises the error, if you input non-digit ID, except block immediately catches the error, and handles it by showing a message.

“finally” statement is always executed at the end of the program, displays a message or does any other tasks.

Debug

“assert” statement can add error-checking code to the script, to check the error.

```
assert   (test-expression), error-message
```

“test-expression” looks like “a>2”, “b!=3”…….it returns true or false.

In assert statement, if test-expression returns false, an error message will appear.

Example 7.8

```
myArr = [“a”, “b”, “c”, “d”, “e”]
def show(key):
    assert (key > 5), “Index out of range!”
    print(key)
key = 5
show(key)
```

Output: ……AssertionError: Index out of range!

Explanation:

“assert (key > 5), “Index out of range!”” is an assertion statement, which executes the test expression(key>5) first, if it returns false, the error message “Index out of range” will be shown.

Exercises: Show Other Flowers

Import Module Demo

Please click “Start > Programs > Python3.5 > IDLE (Python GUI)”.

Write the following code to IDLE editor:

```
# program001.py
def red():
    print (“This flower is red”)
def yellow():
    print (“This flower is yellow”)
def green():
   print (“This flower is green”)
```

Save the file named **program001.py**, and close the file.

“program001.py” should be in the same directory with the following working file.

Please click “Start > Programs > Python3.5 > IDLE (Python GUI)”.

Write the following code to IDLE editor:

```
# program002.py
import program001
program001.red()
program001.yellow()
program001.green()
```

Save the file named **program002.py**, and run the program by pressing **F5,** (Run > Run Module).

Output:

```
This flower is red

This flower is yellow

This flower is green
```

Exercises: Handle Error

Exception Demo

Please click “Start > Programs > Python3.5 > IDLE (Python GUI)”.

Write the following code to IDLE editor:

```
try:
    open(“noFile.txt”, “r”)
except:
    print(“Exception occurs! The file is not found.”)
```

Save the file, and run the program by pressing **F5** key.

(Run > Run Module).

Output:

```
Exception occurs! The file is not found.
```

Hour 8

Class & Object

Class Definition

A class is a template for object, and creates an object.

```
class ClassName:       # define a class
    classVariable = value          # declare a class variable
    def  __init__(self):        # declare a constructor,
    def classMethod(self):      # define a class method
```

__init__(self) is a constructor for initialization. It is automatically called when an object is created.

“self” is a variable that refers to the current object.

“def classMethod(self):” define a class method with argument “self”

The properties of a class are known as “members”.

Example 8.1

```
class Animal:        # define a class
count = 0
def __init__(self):
    self.name = value 1
    self.size = value2
def show(self):
    print (self.name)
    print (self.size)
```

Explanation:

class Animal: creates a class “Animal”,

count = 0 declares a class variable.

def __init__(self)**:** defines an constructor,

constructor is used to initialize the variables.

def show(self): defines an class method

Note:

The first letter of class name should be uppercase.

“self” is used to reference a variable or method.

For example:

self.variable

self.method()

Object Declaration

Object is an instance of a class.

```
objectName = ClassName( args )    # create an object
```

Example 8.2

```
class Animal:     # define a class Animal
    count = 88     # declare a class variable
    def __init__(self, value1, value2):   # define a constructor
    self.name = value1
    self.age = value2
    def show(self):     # define a class method
    print (“The animal name is ” + self.name)
    print (“The tiger age is “+ self.age)
tiger = Animal(“Tiger”, “100”)    # create an object
tiger.show()     # object references method
print (“Tiger counts ” + str(tiger.count))   # object references variable
```

Output:

```
The animal name is Tiger

The tiger age is 100

Tiger counts 88
```

Explanation:

“tiger = Animal(“Tiger”, “100”)” creates an object “tiger”, automatically calls constructor __init__ (self, value1, value2), and pass two arguments “Tiger”, “100” to this constructor,

self.name = Tiger,   self.age = 100

“self” represents the current object “tiger”.

“tiger.show()” means the object “tiger” reference method “def show(self)”.

“str()” changes data type from number to string.

Please save the file named “Animal.py”.

Another Object

```
from anotherFile import*        # import anything from another file

obj = className(args)      # create a new object
```

Example 8.3

```
from Animal import*

cat = Animal(“Meo”, “10”)        # create an object cat

print (“The name of the cat: ” + cat.name)

print (“The age of the cat: ” + str(cat.age))
```

Output:

```
The animal name is Tiger

The tiger age is 100

Tiger counts 88

The name of the cat: Meo

The age of the cat: 10
```

Explanation:

“from Animal import*” imports anything from Animal.py.

“cat = Animal(“Meo”, “10”)” creates an object “cat”, and constructor __init__(self) will be automatically called when an object is created.

Constructor __init__(self) initializes the variables:

“self.name = “Meo””

“self.age = 10”

“self” represents the current object “cat”.

str( ) changes the data type as string.

Because of “from Animal import*”, the output includes the contents from Animal.py.

Inheritance

A Python class can be derived a new sub class. The sub (derived) class inherits all members of the parent (base) class.

```
class BaseClass:     # define a base class
......
class DerivedClass (BaseClass):     # define a derived class
......
```

Example 8.4

```
class Computer:     # define a base class
    harddrive = 10000
    memory = 8

    def setValue(self, harddrive, memory):   # base method
    Computer.harddrive = harddrive
    Computer.memory = memory

class Desktop(Computer):     # define a derived class
    def capacity(self):        # derived method
    print (“Desktop”)
    print (“Harddrive capacity: ” + self.harddrive)
    print (“Memory capacity: ” + self.memory)

d = Desktop()     # create an object
d.setValue(“9000”, “7”)     # initialize value
d.capacity()      # call a derived method
```

Output:

```
Desktop

Harddrive capacity: 9000

Memory capacity: 7
```

Explanation:

“class Desktop(Computer):” means that a derived class Desktop inherits the base class Computer and its members.

The members of the derived class inherit the members of the base class.

“self” represents the object of derived class.

Overriding Method

When a method name in derived class is the same as the method name in base class, it is known as “overriding base method”

```
class BaseClass

def methodName():     # base method

……

class DerivedClass(BaseClass):

def methodName():      # derived method

……
```

Because the derived method name is the same as the base method name, the derived method will override the base method, and executes the derived method instead of base method.

Example 8.5

```
class Computer:      # define a base class
    def __init__(self, name):   # define a constructor
    self.name = name
    def capacity(self, harddrive, memory):    # base method
    self.harddrive = harddrive
    self.memory = memory

class Laptop(Computer):     # define a derived class
    def capacity(self, harddrive, memory):    # derived method
    print self.name
    print (“Harddrive capacity: ” + harddrive)
    print (“Memory capacity: “+ memory)

l = Laptop(“Laptop”)       # creates an object “l”
l.capacity(“8000”, “6”)      # call capacity( )
```

Output:

```
Laptop

Harddrive capacity: 8000

Memory capacity: 6
```

Explanation:

l.capacity(“8000”, “6”) calls the method capacity(), because the derived method name is the same as the base method name at this time, the derived method overrides the base method, executes the derived method instead of base method, and print out the result.

“Overriding” always happens between base class and derived class.

Polymorphism

Polymorphism describes the ability to perform different method for different object if a program has one more classes.

Example 8.6

```
class Dog:              # define a class
    def cry(self):      # define a cry() method
    print ("Dog cries: Wou! Wou!")
class Cat:              # define a class
    def cry(self):      # define a cry() method
    print ("Cat cries: Meo! Meo!")
d = Dog()
d.cry()
c = Cat()
c.cry()
```

Output:

```
Dog cries: Wou! Wou!

Cat cries: Meo! Meo!
```

Explanation:

“d.cry()” calls the cry() method in class Dog.

“c.cry()” calls the cry() method in class Cat.

Exercises: Display Name & …

Class & Object

Please click “Start > Programs > Python3.5 > IDLE (Python GUI)”.

Write the following code to IDLE editor:

```
class Flower:
    def __init__(self, name, color ):
    self.name = name
    self.color = color
f = Flower(“rose”, “red”)
print (“The flower’s name is ” + f.name)
print (“The flower’s color is ” + f.color)
```

Save the file, and run the program by pressing **F5** key.

Output:

```
The flower’s name is rose

The flower’s color is red
```

Appendix

Python Charts

List Functions

Functions	**Operation**
list.append(n)	Append n to the end of list
list.count(n)	Count how many n
list.index(n)	Return the index of n
list.insert(i,n)	Insert n before index i
list.pop(i)	Remove & return the item at index i
list.remove(n)	Remove the n
list.reverse()	Reverse the sequence of list
list.sort()	Sort the element of list increasingly
list extend(lst)	Append each item of lst to list

Tuple Functions

Functions	Operation
x in tpl	return true if x is in the tuple
len(tpl)	return length of tuple
tpl.count(x)	count how many x in tuple
tpl.index(x)	return the index of x

Set Functions

Functions	Operation
set.add(n)	Add x to the set
set.update(a, b, c)	Add a, b, c to the set
set.copy( )	Copy the set
set.remove(n)	Remove the item n
set.pop( )	Remove one random item
set1.intersection(set2)	Return items in both sets
set1.difference(set2)	Return items in set1 not in set2

Dictionary Functions

Functions	**Operation**
d.items( )	return key**:**value pairs of d
d.keys()	return keys of d
d.values()	return values of d
d.get(key)	return the values with specified key
d.pop(key)	remove key and return its value
d.clear()	remove all items of d
d.copy()	copy all items of d
d.setdefault(k,v)	set key**:**value to d
d1.update(d2)	add key**:**value in d1 to d2

Difference

Structures	**Descriptions**
List	store multiple changeable values
Tuple	store multiple unchangeable values
Set	store multiple unique values
Dictionary	store multiple key**:**value pairs

String Operating

Operators	Description
+	concatenate strings together
*	repeat a string
[key]	return a character of the string
[key1: key2]	return characters from key1 to key2-1
in	check a character existing in a string
not in	check a character not existing in a string
’’’ ’’’	describe a function, class, method…

Escape

Characters	Description
\	escape backslash
\’	escape single quote
\”	escape double quote
\n	new line
\r	return
\t	tab

Testing Functions

Functions	**Return True if…**
isalpha()	return true if all characters are letters
isdigit()	return true if all characters are digits
isdecimal()	return true if all characters are decimals
isalnum()	return true if all characters are numbers or letters
islower()	return true if all characters are lowercase
isupper()	return true if all characters are uppercase
istitle()	return true if the string is title-case string
isspace()	return true if the string contains only whitespace

Searching Functions

Functions	Return
find(c)	return the index of first occurrence, or -1
rfind(c)	same as find(), but find from right to left
index(c)	return the index of first occurrence
rindex(c)	same as index(), but find from right to left

Formating Functions

Functions	Returned String
center(w, f)	center string with width w and fill with f
ljust(w,f)	left adjust string with width w and fill with f
rjust(w,f)	right adjust string with width w and fill with f

Stripping Functions

Functions	Returned String
strip()	remove leading and trailing spaces
lstrip()	remove leading spaces
rstrip()	remove trailing spaces

Splitting Functions

Functions	Returned String
split(separator)	split a string by a separator. (default whitespace as a separator)
partition(separator)	partition a string by a separator to three parts. (head, separator, tail)

Strings Functions

Functions	Returned Strings
replace(old, new)	replace every old with new
count(ch)	count the number of the characters
capitalize()	change the first letter to uppercase
separater.join()	join the strings by separator
str.swapcase()	swap the letters case of the string
str.zfill(length)	add zeros to the left of the string with length

Regular Expression

Operators	Matches
^	Matches beginning of line.
$	Matches end of line.
.	Matches any single character.
[…]	Matches any single character in brackets.
[^…]	Matches any single character not in brackets
?	Matches 0 or 1 occurrence
+	Matches 1 or more occurrence
*	Matches 0 or more occurrences
{ n}	Matches exactly n number of occurrences
{ n, m}	Matches at least n and at most m occurrences
a | b	Matches either a or b.
(re)	Groups regular expressions
\w	Matcheses word characters.
\W	Matcheses non-word characters.
\s	Matcheses space.
\S	Matcheses non-space.
\d	Matcheses numbers.

\D	Matcheses non-numbers.

File Methods

File methods:

file = open (*filename,mode*)

file.readable()

file.writable()

file.read(size)

file.readlines(size)

file.seek(offset)

file.write(string)

file.close()

File Modes

modes	**actions**
r	open file for reading (default)
w	open file for writing
a	open file for appending
+	open file for reading & writing
b	open file in binary mode
t	open file in text mode

How time flies! It is time to say good-bye.

Source Code for Download

Please download the source code of this book:

Python Source Code for Download

Dear My Friend,

If you are not satisfied with this eBook, could you please return the eBook for a refund within seven days of the date of purchase?

If you found any errors in this book, could you please send an email to me?

yaowining@yahoo.com

I will appreciate your email. Thank you very much!

Sincerely

Ray Yao

My friend, See you!