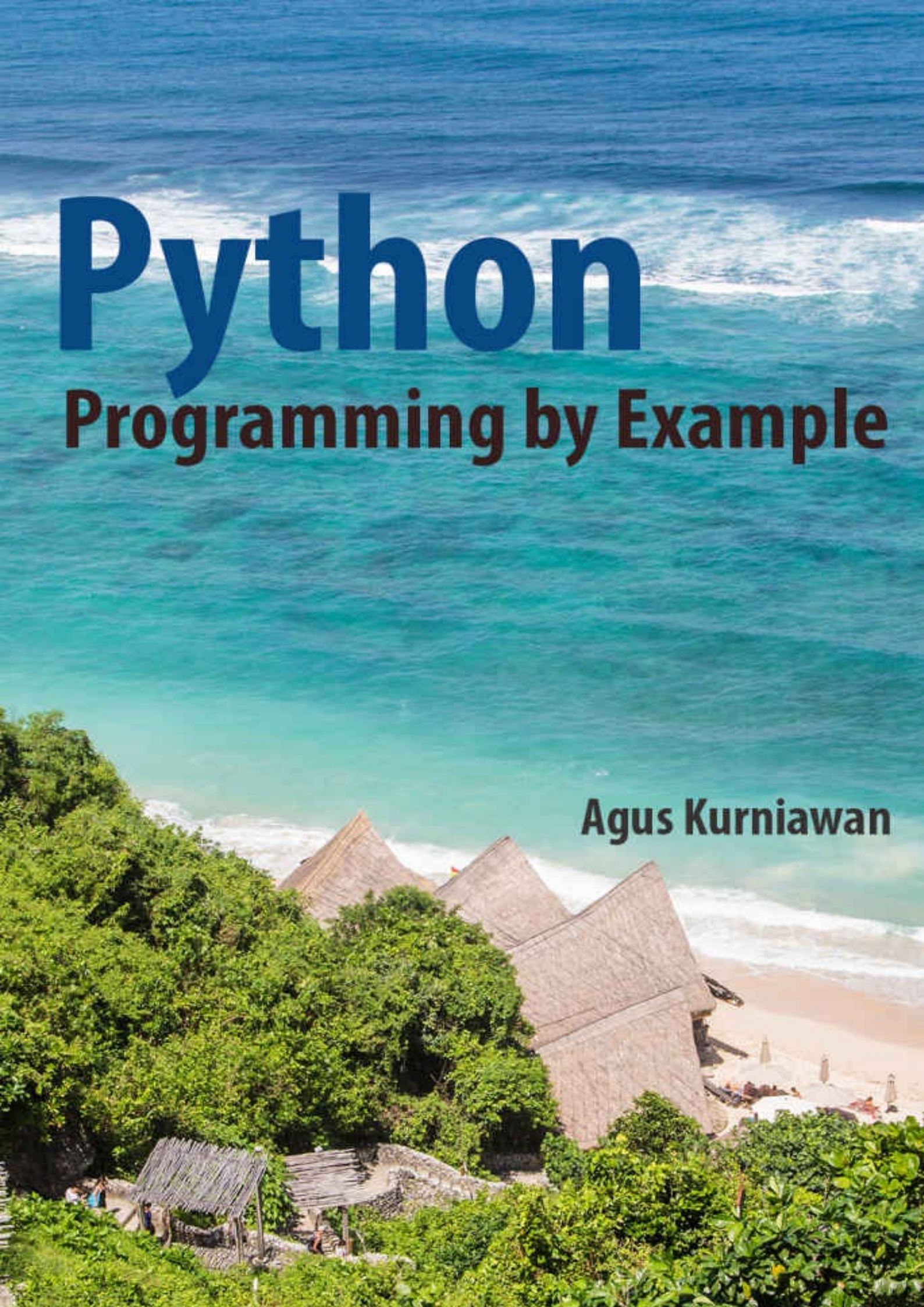
Python

Programming by Example



Agus Kurniawan

Copyright

Python Programming by Example

Agus Kurniawan

1st Edition, 2015

Copyright © 2015 Agus Kurniawan

* Cover photo is credit to Fajar Ramadhany, Bataviasoft, http://bataviasoft.com/.

Table of Contents

Copyright

Preface

1. Development Environment

1.1 Installation

1.2 Development Tools

1.3 Python Shell

1.4 Running Python Application from Files

2. Python Programming Language

2.1 Common Rule

2.2 Variables

2.3 Comment

2.4 Arithmetic Operations

2.5 Mathematical Functions

2.6 Increment and Decrement

2.7 Getting Input from Keyboard

2.8 Python Operators

2.8.1 Comparison Operators

2.8.2 Logical Operators

2.8.3 Bitwise Opeators

2.8.4 Testing All

2.9 Decision Control

2.10 Iteration - for and while

2.11 break, continue and pass

2.12 Date & Time

3. Lists, Tuples and Dictionary

3.1 Lists

3.2 Tuples

3.3 Dictionary

4. Functions

4.1 Creating A Simple Function

4.2 Function with Parameters and Returning Value

4.3 Function with Multiple Returning Values

4.4 Recursion Function

4.5 Testing

5. Python Object Oriented

5.1 Creating Classes

5.2 Class Attributes

5.3 Built-In Class Attributes

5.4 Destroying Class Object

5.5 Write them All

5.6 Inheritance

5.7 Overriding Methods

5.8 Overloading Operators

6. Python Modules and Packages

6.1 Python Modules

6.2 import Statement

6.3 from…import * Statement

6.4 Installing External Python Package

7. String Operations

7.1 Getting Started

7.2 Concatenating Strings

7.3 String To Numeric

7.4 Numeric to String

7.5 String Parser

7.6 Check String Data Length

7.7 Copy Data

7.8 Upper and Lower Case Characters

7.9 Testing A Program

8. File Operations

8.1 Getting Started

8.2 Writing Data Into A File

8.2.1 Creating a File

8.2.2 Writing Data

8.2.3 Closing a File

8.2.4 Demo

8.3 Reading Data From A File

9. Error Handling

9.1 Error Handling

9.2 Catching All Errors

9.3 Raising Exceptions

9.4 Custom Exception

10. Building Own Python Module

10.1 Creating Simple Module

10.2 Building Own Python Package

11. Concurrency

11.1 Getting Started

11.2 Threading

11.3 Synchronizing Threads

11.3.1 Mutex Locks

11.3.2 Event

11.4 Queue

11.5 Multiprocessing

11.5.1 Process

11.5.2 Synchronizing Processes

11.6 Parallel Tasks

11.6.1 ThreadPoolExecutor

11.6.2 ProcessPoolExecutor

12. Encoding

12.1 Getting Started

12.2 Encoding Base64

12.3 Hexadecimal

12.4 JSON

12.5 XML

12.6 CSV

13. Hashing and Cryptography

13.1 Getting Started

13.2 Hashing

13.2.1 Hashing with MD5

13.2.2 Hashing with SHA1 and SHA256

13.2.3 Hashing with Key Using HMAC

13.2.4 Write them All

13.3 Cryptography

13.3.1 Symmetric Cryptography

13.3.2 Asymmetric Cryptography

14. Database Programming

14.1 Database for Python

14.2 MySQL Driver for Python

14.3 Testing Connection

14.4 CRUD (Create, Read, Update and Delete) Operations

14.4.1 Create Data

14.4.2 Read Data

14.4.3 Update Data

14.4.4 Delete Data

14.4.5 Write them All

15. Socket Programming

15.1 Socket Module

15.2 Hello World

15.3 Client/Server Socket

15.3.1 Server Socket

15.3.2 Client Socket

15.3.3 Testing

16. Python Regular Expressions

16.1 Getting Started

16.2 Demo

17. Python GUI Programming

17.1 Getting Started

17.2 Hello Python GUI

17.3 Working with Input Form

17.4 Working with Common Dialogs

18. Python Unit Testing

18.1 Getting Started

18.2 Demo

Source Code

Contact

Preface

This book is a brief reference to the Python programming language. It describes all the elements of the language and illustrates their use with code examples.

Agus Kurniawan

Depok, November 2015

1. Development Environment

1.1 Installation

Python is a widely used general-purpose, high-level programming language. Installation of Python application is easy. For Windows, Linux and Mac Platform, you download setup file from Python website, https://www.python.org/downloads/. Download and run it. Follow installation commands.

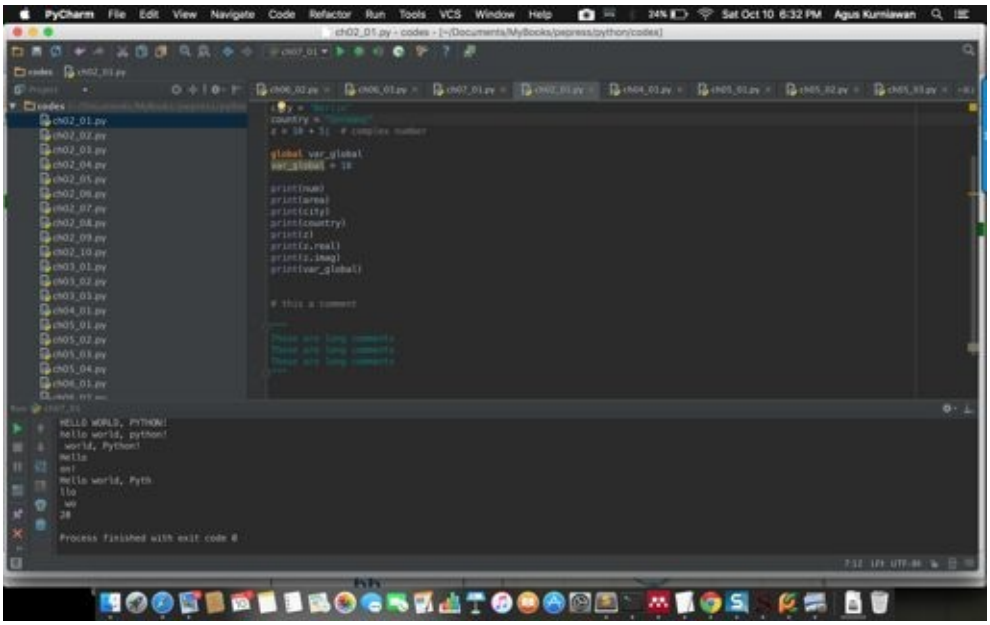
If you're working on Windows platform, you can run setup file and follow instruction.

1.2 Development Tools

Basically, you can use any text editor to write Python code. The following is a list of text editor:

- vim
- nano
- PyCharm, https://www.jetbrains.com/pycharm/
- Intellij IDEA, https://www.jetbrains.com/idea/
- Sublime text, http://www.sublimetext.com/
- Visual Studio, https://www.visualstudio.com

In this book, I use PyCharm for development tool. Jetbrains provides community and Education licenses for PyCharm.

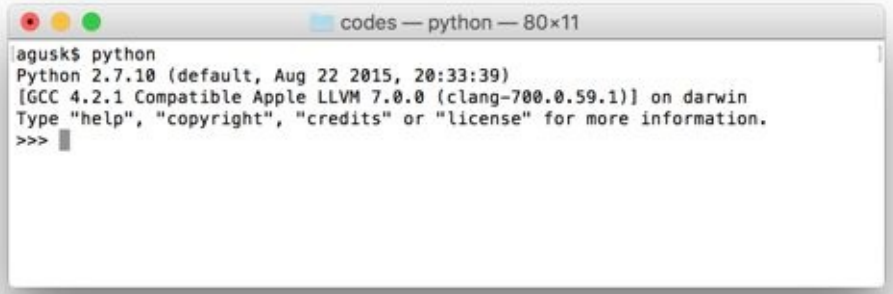


1.3 Python Shell

After installed Python, you obtain Python shell on your platform. You can type this command on Terminal or Command Prompt for Windows Platform.

```
python
```

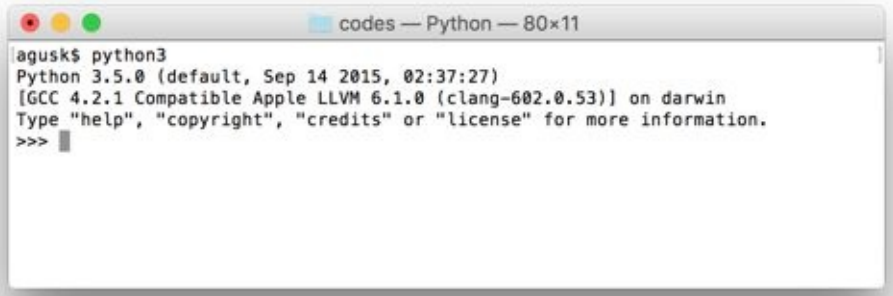
This is Python 2.x. Then, you get Python shell, shown in Figure below.



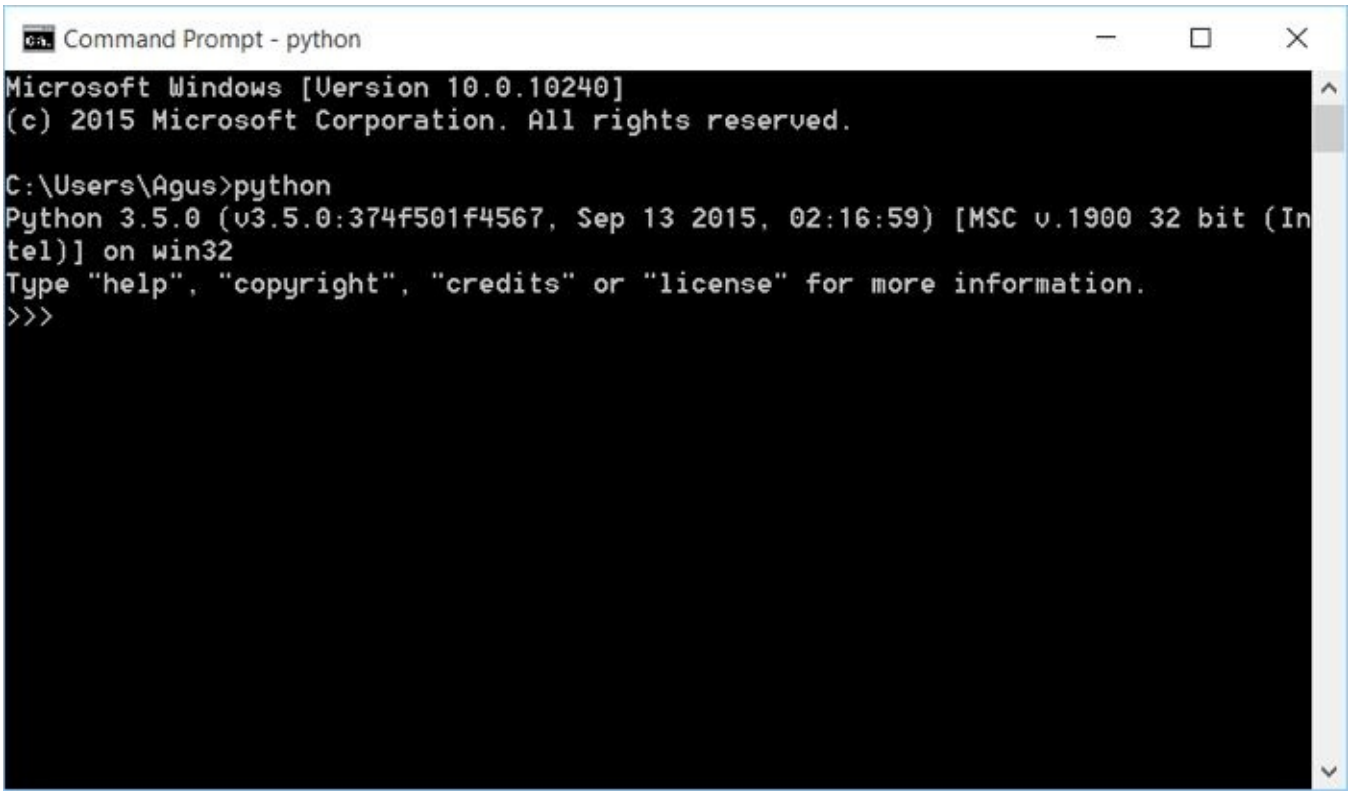
If you installed Python 3.x, you can Python shell by typing this command.

```
python3
```

This is Python 3.x. Then, you get Python shell, shown in Figure below.



Output program on Windows platform.



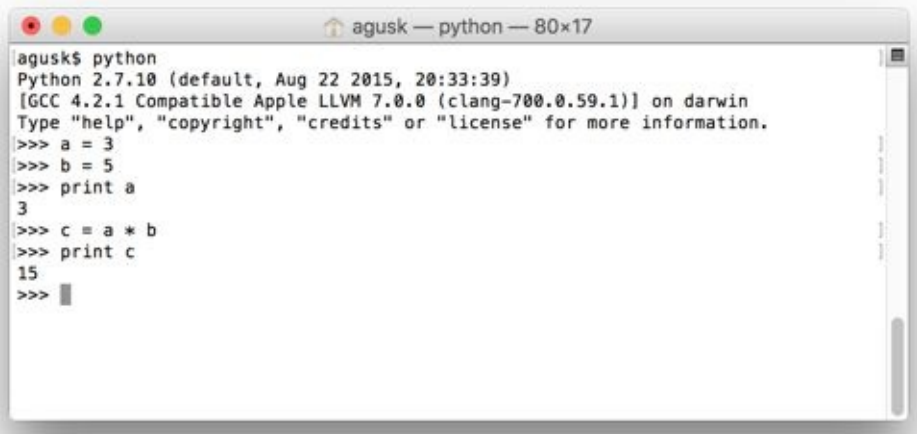
After you call Python shell, you obtain the shell. It shows >>> on Terminal.

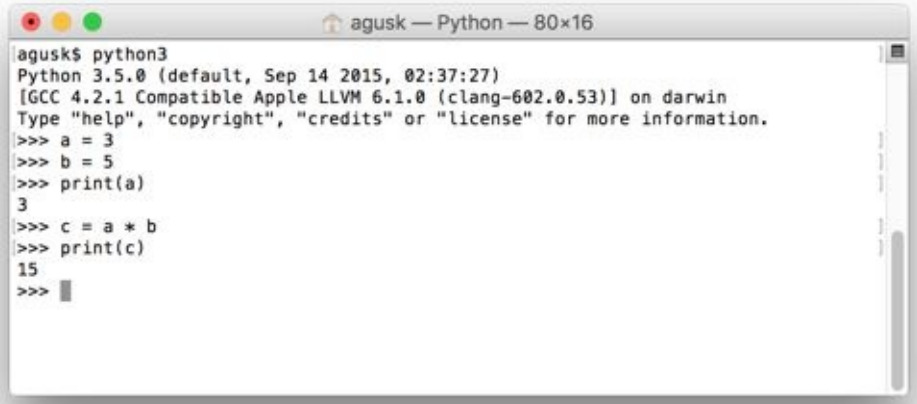
Try to do the following command.

```
>>> a = 3
>>> b = 5
>>> print a
>>> c = a * b
>>> print c
```

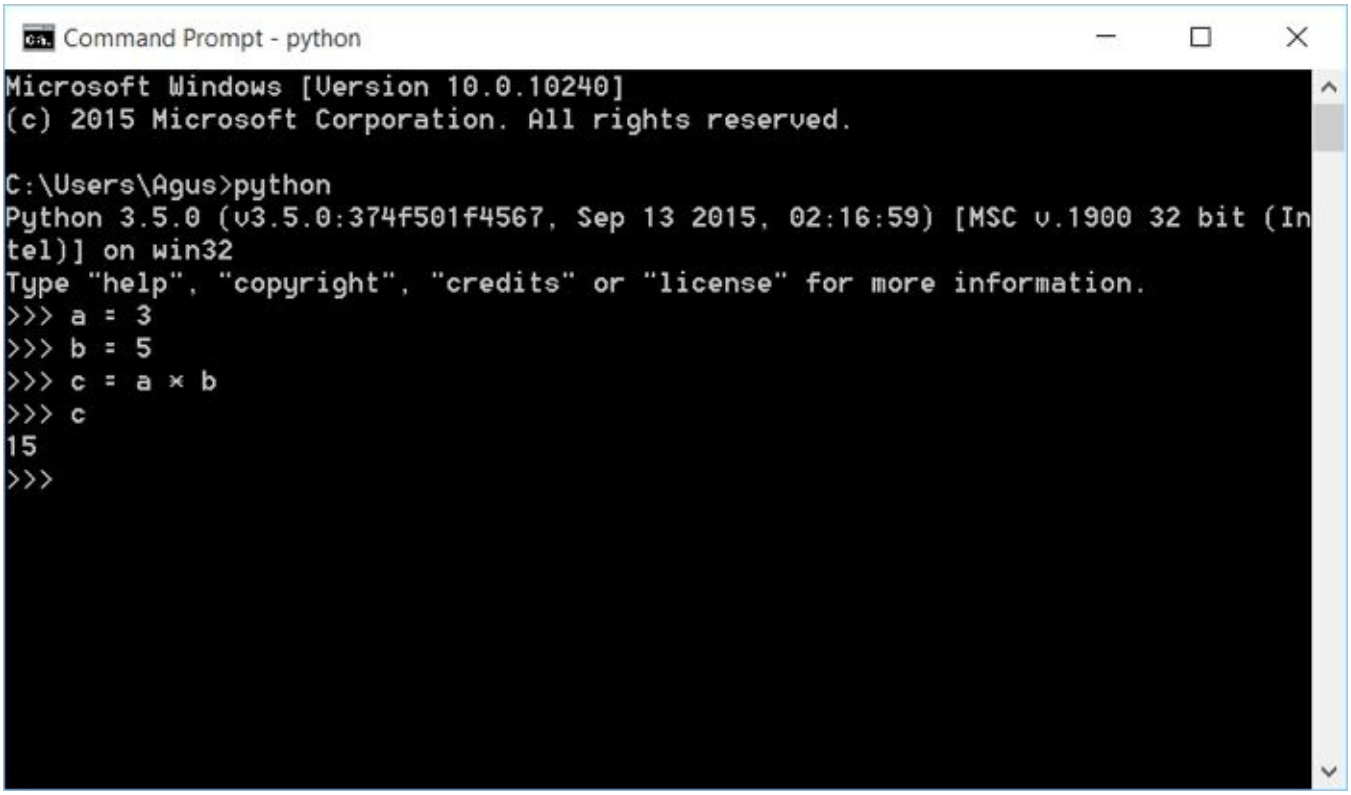
In Python 3.x, print a is replaced by print(a).

The following is a sample output of program.





A sample output for Windows platform.



1.4 Running Python Application from Files

You can run your program by writing them on a file. For instance, you create a file, called ch01_01.py, and write this script.

```
print('hello world from python')
```

To run the program, you can type this command on Terminal.

```
python ch01_01.py
```

If you want to run the program under Python 3.x, type this command on Terminal.

```
python3 ch01_01.py
```

Program output:

2. Python Programming Language

This chapter explains the basic of Python programming language.

2.1 Common Rule

Python language doesn’t write “;” at the end of syntax like you do it on C/C++ languages. Here is the syntax rule of Python:

```
syntax_code1
syntax_code2
syntax_code3
```

2.2 Variables

In this section, we explore how to define a variable and assign its value. By default, we define variables on Python with assigning value .

```
# declare variables
num = 2
area = 58.7
city = 'Berlin'
country = "Germany"
z = 10 + 5j  # complex number
```

If you want to declare variables without assigning values, you can set it using None.

```
# declare variable without initializing value
counter = None
index = None
```

Write these codes for testing.

```
# declare variables
num = 2
area = 58.7
city = 'Berlin'
country = "Germany"
z = 10 + 5j  # complex number

# declare variable without initializing value
counter = None
index = None


global var_global
var_global = 10

print(num)
print(area)
print(city)
print(country)
print(z)
print(z.real)
print(z.imag)
print(var_global)
```

Save these scripts into a file, called ch02_01.py.

Now you can type this file using Python 3.x.

```
$ python3 ch02_01.py
```

A sample of program output can seen in Figure below.

2.3 Comment

You may explain how to work on your code with writing comments. To do it, you can use # and """ syntax. Here is sample code:

```
# this a comment

"""
These are long comments
These are long comments
These are long comments
"""
```

2.4 Arithmetic Operations

Python supports the same four basic arithmetic operations such as addition, subtraction, multiplication, and division. For testing, create a file, called **ch02_02.py**.

The following is the code illustration for basic arithmetic in **ch02_02.py**:

```
a = 2.3
b = 8

c = a + b
print(c)
c = a - b
print(c)
c = a * b
print(c)
c = a / b
print(c)
```

Save and run this program.

```
python3 ch02_02.py
```

A sample of program output:

2.5 Mathematical Functions

Python provides math library. If you're working with Python 2.x, you can read this library on https://docs.python.org/2/library/math.html . For Python 3.x, you can read math library on https://docs.python.org/3/library/math.html .

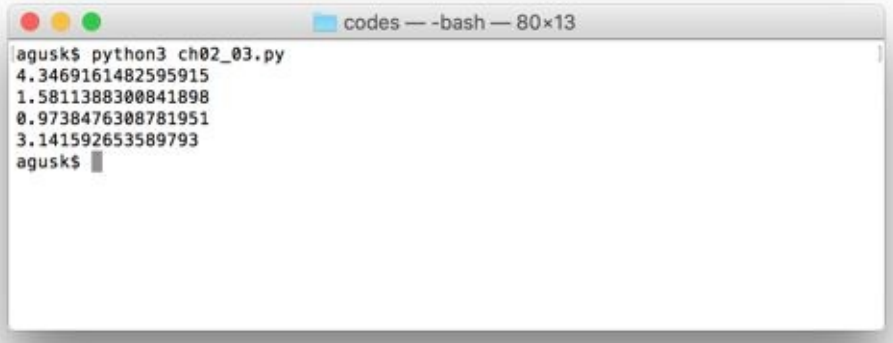
Create a file, called **ch02_03.py**. Write the following code.

```
from math import *

a = 1.8
b = 2.5

c = pow(a, b)
print(c)

c = sqrt(b)
print(c)

c = sin(a)
print(c)

print(pi)
```

Save and run the program.

```
python3 ch02_03.py
```

A sample of program output:



2.6 Increment and Decrement

Python doesn't has special syntax for increment and decrement. We can define increment and decrement as follows.

- ++ syntax for increment. a++ can be defined as a = a + 1
- — syntax for decrement. a— can be defined as a = a - 1

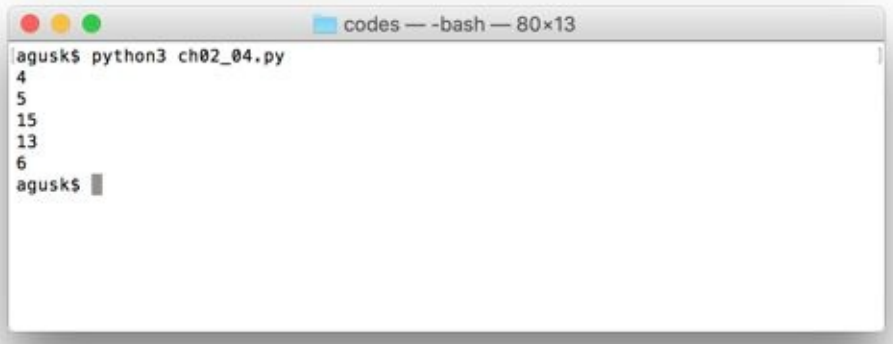
For testing, create a file, called **ch02_04.py**. Write the following script.

```
a = 4
print(a)

# increment
a = a + 1
print(a)

a += 10
print(a)

# decrement
a = a - 2
print(a)

a -= 7
print(a)
```

Then, save and run the program.

```
python3 ch02_04.py
```

A sample of program output:



2.7 Getting Input from Keyboard

To get input from keyboard, we can use input() for Python 3.x and raw_input() for Python 2.x.

For testing, create a file, called **ch02_05.py**, and write this script.

```
# getting input from keyboard using input()
name = input('What is your name?')
print('Hello,' + name + '!!')
user_id = input('What is your ID?')
print('Id is ' + str(user_id))


# getting input from keyboard using raw_input()
product = raw_input('Product name?')
print('Product=' + product)
product_id = raw_input('Product ID?')
print('Product id= ' + str(product_id))
```

Save and run the program.

```
python3 ch02_05.py
```

A sample of program output:

2.8 Python Operators

In this section, we learn several Python operators such as comparison, logical and bitwise operators.

2.8.1 Comparison Operators

You may determine equality or difference among variables or values. Here is the list of comparison operators:

```
==       is equal to
!=       is not equal
>        is greater than
<        is less than
>=       is greater than or equal to
<=       is less than or equal to
```

2.8.2 Logical Operators

These operators can be used to determine the logic between variables or values.

```
&&      and
||      or
!        not
```

2.8.3 Bitwise Opeators

Bitwise operators in Python can be defined as follows

```
&     AND
|   OR
^   Exclusive OR
>>    Shift right
<<    Shift left
~   Not (Inversion)
```

2.8.4 Testing All

Now we test how to use comparison, logical and bitwise operators in code. Create a file, called **ch02_06.py**, and write these scripts.

```
# comparison operators
a = 3
b = 8

print(a == b)
print(a != b)
print(a > b)
print(a >= b)
print(a < b)
print(a <= b)


# logical operators

print((a == b) and (a != b))
print((a <= b) or (a > b))
print(not (a >= b))


# bitwise operators
# declare binary variables
m = 0b01010011
n = 0b11111001

print(m)
print(n)
print(bin(m & n))
print(bin(m | n))
print(bin(m ^ n))
print(bin(~m))
print(bin(b << 3))
print(bin(b >> 2))
```

Save this file and run the program.

```
python ch02_06.py
```

A sample of program output:

codes — -bash — 80×19
agusk$ python3 ch02_06.py
False
True
False
False
True
True
False
True
True
83
249
0b1010001
0b11111011
0b10101010
-0b1010100
0b1000000
0b10
agusk$

2.9 Decision Control

Syntax model for *if..else* can be formulated as below:

```
if (conditional):
  # do something
else:
 # do something

#########

if (conditional):
  # do something
elif (conditional):
 # do something
else:
 # do something
```

For testing, create a file, called **ch02_07.py** and write these scripts.

```
# if-else
a = 10
b = 30

print('demo if-elif-else')
if (a > 10) or (b > 10):
    # do something
    print('(a > 10) or (b > 10)')

elif (a != 5) and (b <= 7):
    # do something
    print('(a != 5) and (b <= 7)')

else:
    # do something
    print('else')


# nested if
if (a == 0) or (b > 20):
    if b < 50:
        print('nested-if')
    else:
        print('else-nested-if')
else:
    print('if-else')
```

Save and run the program.

```
python3 ch02_07.py
```

A sample of program output:

2.10 Iteration - for and while

Iteration operation is useful when we do repetitive activities. The following is for syntax.

```
for (iteration):
 # do something


while (conditional):
  # do something
```

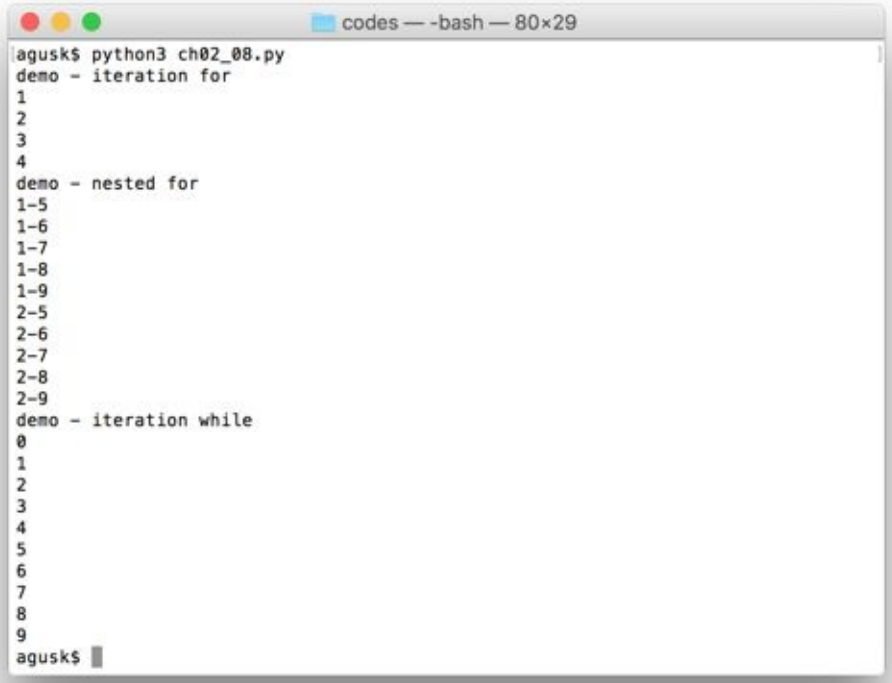
For testing, create a file, called **ch02_08.py** and write these scripts.

```
# iteration - for
print('demo - iteration for')
for i in range(1, 5):
    print(i)


# nested - for
print('demo - nested for')
for i in range(1, 3):
    for j in range(5, 10):
        print(str(i) + '-' + str(j))


# iteration - while
print('demo - iteration while')
i = 0
while i < 10:
    print(i)
    i += 1
```

Save and run the program

```
python3 ch02_08.py
```

A sample of program output:


codes — -bash — 80×29
agusk$ python3 ch02_08.py
demo - iteration for
1
2
3
4
demo - nested for
1-5
1-6
1-7
1-8
1-9
2-5
2-6
2-7
2-8
2-9
demo - iteration while
0
1
2
3
4
5
6
7
8
9
agusk$

2.11 break, continue and pass

break can be used to stop on the code point. Otherwise, continue can be used to skip some scripts. For illustration, we have a looping. The looping will be stopped using *break* if value = 7. Another sample, we can skip the iteration with value = 4 using *continue* syntax.

Write these scripts.

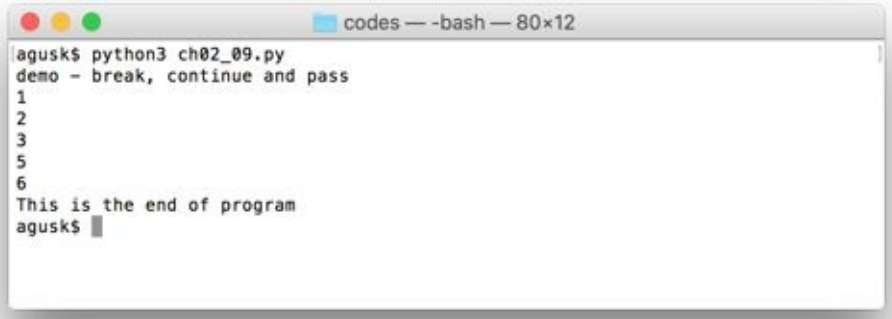
```
print('demo - break, continue and pass')
for i in range(1, 10):
    if i == 4:
        continue

    if i == 7:
        break
    print(i)


pass  # do nothing
print('This is the end of program')
```

Save these scripts into a file, called **ch02_09.py**.

Run the program.

```
python3 ch02_09.py
```

A sample of program output:



2.12 Date & Time

We can work with Data and time in Python using time library. We must import this library.

For testing, write these scripts.

```
import time

# get current time
now = time.time()  # utc
print(now)
# display readable current time
print(time.strftime("%b %d %Y %H:%M:%S", time.gmtime(now)))
print(time.timezone)
```

Save the program into a file, called **ch02_10.py**.

Run the program.

```
python3 ch02_10.py
```

A sample of program output:

3. Lists, Tuples and Dictionary

This chapter explains how to work with Python collection.

3.1 Lists

Python provides a list for collection manipulation. We define a list as [].

For illustration, we show you how to use a list in Python program. The program implements

- declaring
- printing
- getting a list length
- adding
- getting a specific item from a list
- sorting
- removing

Write these scripts.

```
# declare lists
print('----declare lists')
numbers = []
a = [2, 7, 10, 8]
cities = ['Berlin', 'Seattle', 'Tokyo', 'Moscow']
b = [10, 3, 'Apple', 6, 'Strawberry']
c = range(1, 10, 2)

# print(lists
print('----print(lists')
print(a)
for city in cities:
    print(city)

print(b)
print(c)

# get length of lists
print('----get length of lists')
print(len(a))
print(len(cities))

# add item into list
print('----add item')
numbers.append(10)
numbers.append(5)
cities.append('London')

for i in numbers:
    print(i)

for city in cities:
```

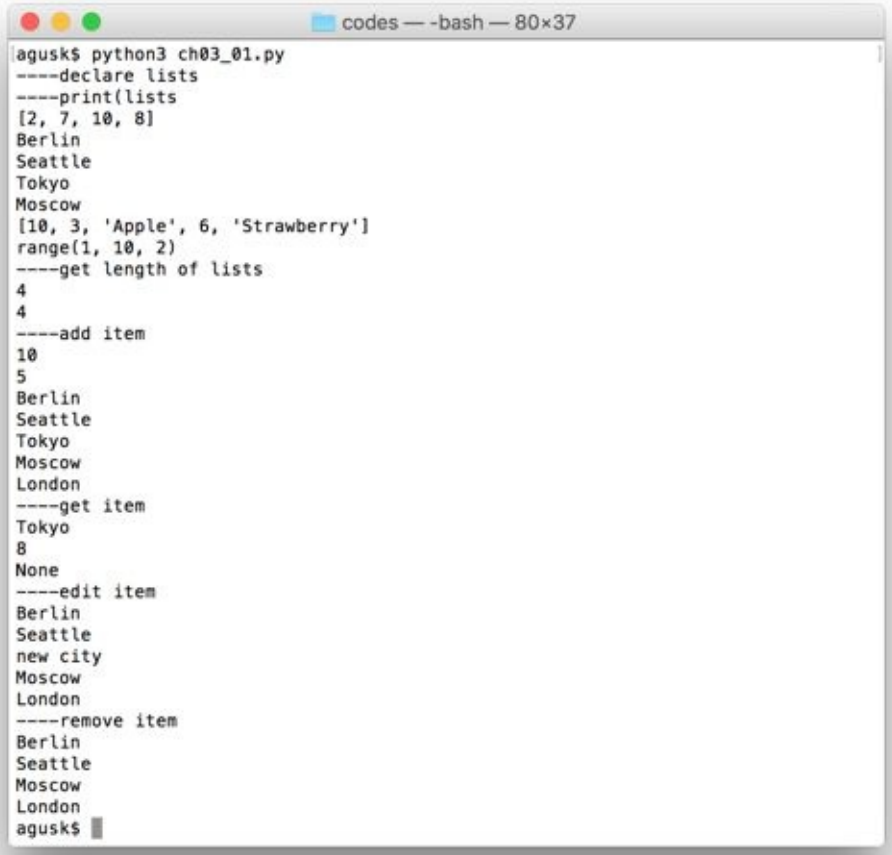
```
    print(city)


# get specific item
print('----get item')
print(cities[2])
print(a[3])


# sorting
print(a.sort())

# edit item
print('----edit item')
cities[2] = 'new city'
for city in cities:
    print(city)


# remove item
print('----remove item')
a.remove(8)   # by value
del cities[2]   # by index
for city in cities:
    print(city)
```

Save thses scripts into a file, called **ch03_01.py**.

Run the program.

```
python3 ch03_01.py
```

A sample of program output:


codes — -bash — 80×37
agusk$ python3 ch03_01.py
----declare lists
----print(lists
[2, 7, 10, 8]
Berlin
Seattle
Tokyo
Moscow
[10, 3, 'Apple', 6, 'Strawberry']
range(1, 10, 2)
----get length of lists
4
4
----add item
10
5
Berlin
Seattle
Tokyo
Moscow
London
----get item
Tokyo
8
None
----edit item
Berlin
Seattle
new city
Moscow
London
----remove item
Berlin
Seattle
Moscow
London
agusk$

3.2 Tuples

We can define a tuple using () in Python. A tuple can be append a new item.

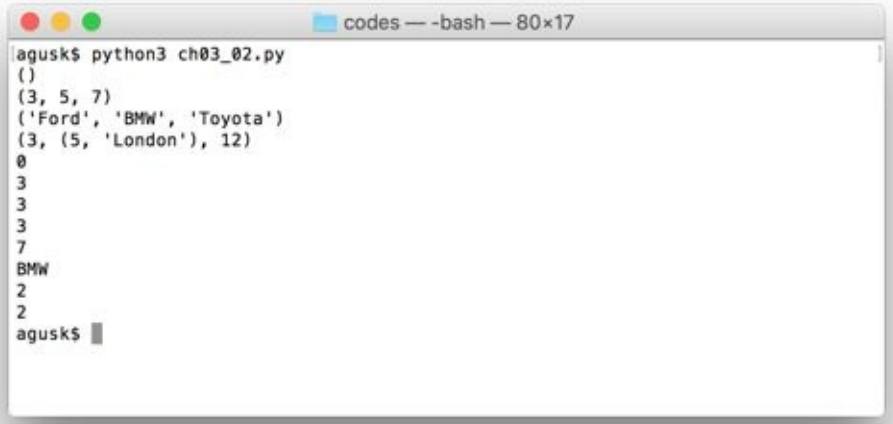
For testing, we build a program into a file, called **ch03_02.py**. Write these scripts.

```
# declare tuples
a = ()
b = (3, 5, 7)
c = ('Ford', 'BMW', 'Toyota')
d = (3, (5, 'London'), 12)

# print
print(a)
print(b)
print(c)
print(d)

# get length of tuples
print(len(a))
print(len(b))
print(len(c))
print(len(d))


# get item
print(b[2])
print(c[1])

# get index
print(b.index(7))
print(c.index('Toyota'))
```

Save and run the program.

```
python3 ch03_02.py
```

A sample of program output:


codes — -bash — 80×17
agusk$ python3 ch03_02.py
()
(3, 5, 7)
('Ford', 'BMW', 'Toyota')
(3, (5, 'London'), 12)
0
3
3
3
7
BMW
2
2
agusk$

3.3 Dictionary

We can create an array with key-value or dictionary. Python uses {} to implement key-value array.

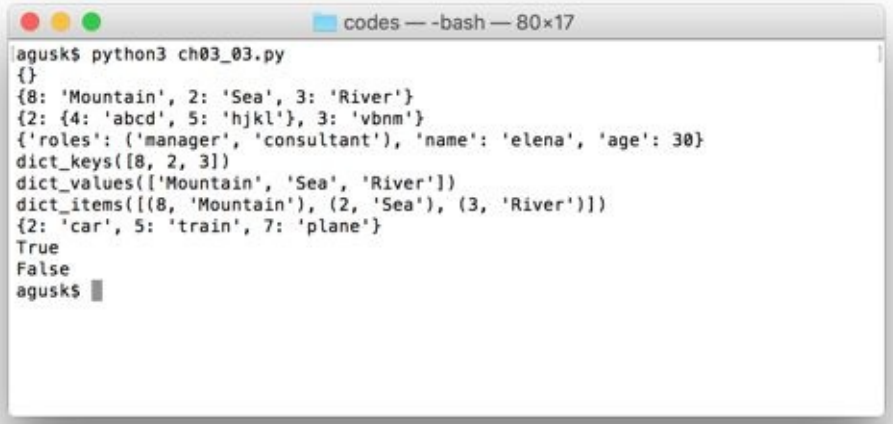
For illustration, we create a program, **ch03_03.py**. Write these scripts.

```
# declare
a = {}
b = {2: 'Sea', 3: 'River', 8: 'Mountain'}
c = {2: {4: 'abcd', 5: 'hjkl'}, 3: 'vbnm'}
d = dict(name='elena', age=30, roles=('manager', 'consultant'))

# print
print(a)
print(b)
print(c)
print(d)

# keys values
print(b.keys())
print(b.values())
print(b.items())

# add item
a.setdefault(2, 'car')
a.setdefault(5, 'train')
a.setdefault(7, 'plane')
print(a)

# check key
print(3 in b)
print(5 in b)
```

Save these scripts and run the program.

```
python3 ch03_03.py
```

A sample of program output:


codes — -bash — 80×17
agusk$ python3 ch03_03.py
{}
{8: 'Mountain', 2: 'Sea', 3: 'River'}
{2: {4: 'abcd', 5: 'hjkl'}, 3: 'vbnm'}
{'roles': ('manager', 'consultant'), 'name': 'elena', 'age': 30}
dict_keys([8, 2, 3])
dict_values(['Mountain', 'Sea', 'River'])
dict_items([(8, 'Mountain'), (2, 'Sea'), (3, 'River')])
{2: 'car', 5: 'train', 7: 'plane'}
True
False
agusk$

4. Functions

This chapter explains how to create function using Python.

4.1 Creating A Simple Function

Declaring function in Python has format as follows.

```
def foo():
    print('foo()')
```

You can call this function foo() on your program.

4.2 Function with Parameters and Returning Value

Sometimes you want to create a function with a parameter. You can implement it as follows.

```
def calculate(val_a, val_b):
    val = val_a * val_b
    return val
```

4.3 Function with Multiple Returning Values

A function can return multiple returning values in Python. For instance, we return three values in Python function. A sample code can be written as below.

```
def perform(num):
    d = num * 5
    return d, d + 5, d - 2
```

4.4 Recursion Function

Recursion function is a function where the solution to a problem depends on solutions to smaller instances of the same problem (as opposed to iteration). For illustration, we can implement Fibonacci problem using Python.

The following is a sample code for Fibonacci solution in Python.

```
def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```

4.5 Testing

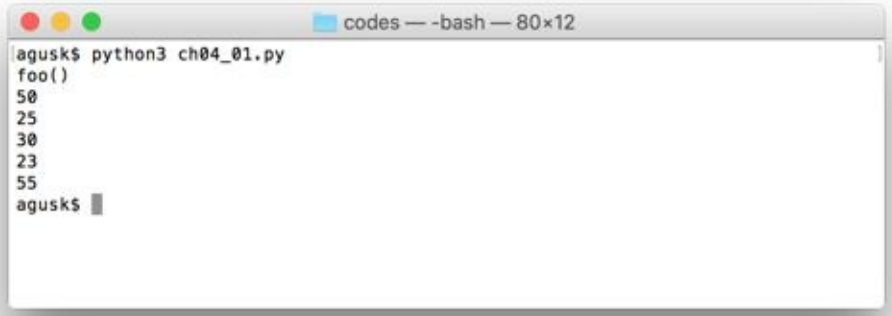
We can write code again from section 4.1 to 4.4 and use them in on **ch04_01.py** file.

```
def foo():
    print('foo()')


def calculate(val_a, val_b):
    val = val_a * val_b
    return val


def perform(num):
    d = num * 5
    return d, d + 5, d - 2


def fibonacci(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n-1) + fibonacci(n-2)


foo()
m = calculate(10, 5)
print(m)
a, b, c = perform(5)
print(a)
print(b)
print(c)
res = fibonacci(10)
print(res)
```

Save and run the program.

```
python3 ch04_01.py
```

A sample of program output:


codes — -bash — 80×12
agusk$ python3 ch04_01.py
foo()
50
25
30
23
55
agusk$

5. Python Object Oriented

This chapter we explore how to work with Object-Oriented programming in Python.

5.1 Creating Classes

Object-oriented programming (OOP) is a programming language model organized around objects. In this chapter, I don’t explain in detail about OOP. I recommend you to read textbooks related to OOP.

In OOP, a class is a template definition of the method s and variable s in a particular kind of object. You can declare a class in Python as follows.

```
from math import *


class City:
    # class data
    city_count = 0
    city_id = 0

   ### do something
```

Note: You can work with OOP using Python 3.x.

After declared a class, we can use it.

```
a = City()
b = City()
```

5.2 Class Attributes

We can declare class attributes to store the data or to communicate to other object.

For instance, we declare city_count and city_id. We also define methods: move_to() and distance().

```
class City:
    # class data
    city_count = 0
    city_id = 0

    # class attributes
    def move_to(self, x=0, y=0):
        self.x += x
        self.y += y

    def distance(self, other_city):
        xi = pow(other_city.x - self.x, 2)
        yi = pow(other_city.y - self.y, 2)

        return sqrt(xi + yi)
```

5.3 Built-In Class Attributes

Basically, Python has built-in class attributes, such as __init__() is used as class contructor and __str__() to generate information about the class.

```
class City:
    # class data
    city_count = 0
    city_id = 0

    # constructor
    def __init__(self, name='', x=0, y=0):
        self.name = name
        self.x = x
        self.y = y
        City.city_count += 1  # access all City classes
        self.city_id = City.city_count

    def __str__(self):
        return 'City: ' + self.name + ',id=' + str(self.city_id) + ',x='
```

5.4 Destroying Class Object

In a class, we can define destructor to clear all usage resouces. We can use __del__() in Python to do that.

```
class City:

    def __del__(self):
        # get class name
        class_name = self.__class__.__name__
        print('class ', class_name, ' destroyed')
```

5.5 Write them All

Let’s try to write the code in implementation. Create a file, called ch05_01.py, and write these scripts.

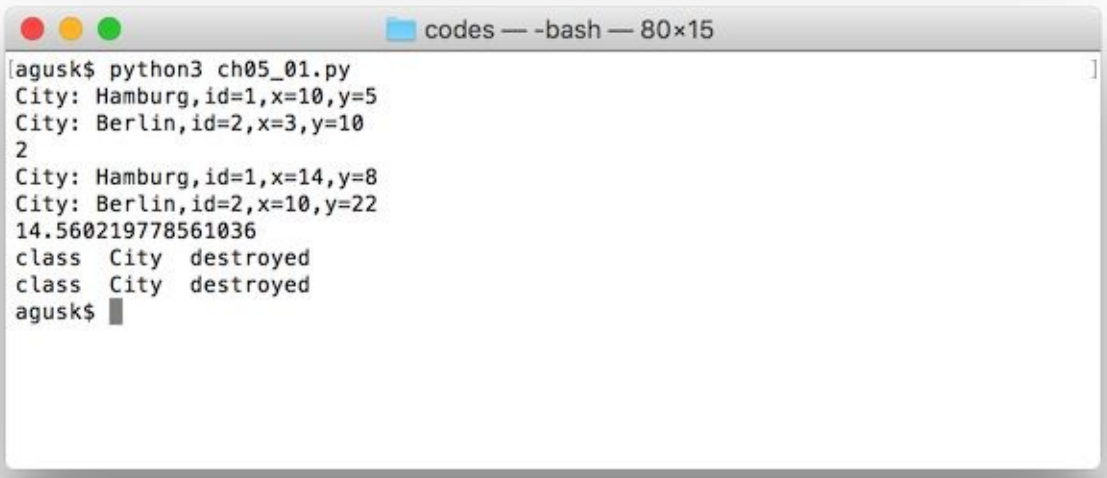
```
from math import *


class City:
    # class data
    city_count = 0
    city_id = 0

    # constructor
    def __init__(self, name='', x=0, y=0):
        self.name = name
        self.x = x
        self.y = y
        City.city_count += 1  # access all City classes
        self.city_id = City.city_count

    def __str__(self):
        return 'City: ' + self.name + ',id=' + str(self.city_id) + ',x='

    # class attributes
    def move_to(self, x=0, y=0):
        self.x += x
        self.y += y

    def distance(self, other_city):
        xi = pow(other_city.x - self.x, 2)
        yi = pow(other_city.y - self.y, 2)

        return sqrt(xi + yi)

    def __del__(self):
        # get class name
        class_name = self.__class__.__name__
        print('class ', class_name, ' destroyed')


a = City('Hamburg', 10, 5)
b = City('Berlin', 3, 10)
print(a)
print(b)
print(City.city_count)


a.move_to(4, 3)
b.move_to(7, 12)
print(a)
print(b)
```

```
distance = a.distance(b)
print(distance)
```

Now you can run the program.

```
$ python3 ch05_01.py
```

A sample of program output:



5.6 Inheritance

Inheritance, encapsulation, abstraction, and polymorphism are four fundamental concepts of object-oriented programming. In this section, we implement inheritance in Python.

Inheritance enables new objects to take on the properties of existing objects. A class that is used as the basis for inheritance is called a superclass or base class. In Python, we can declare inheritance as follows.

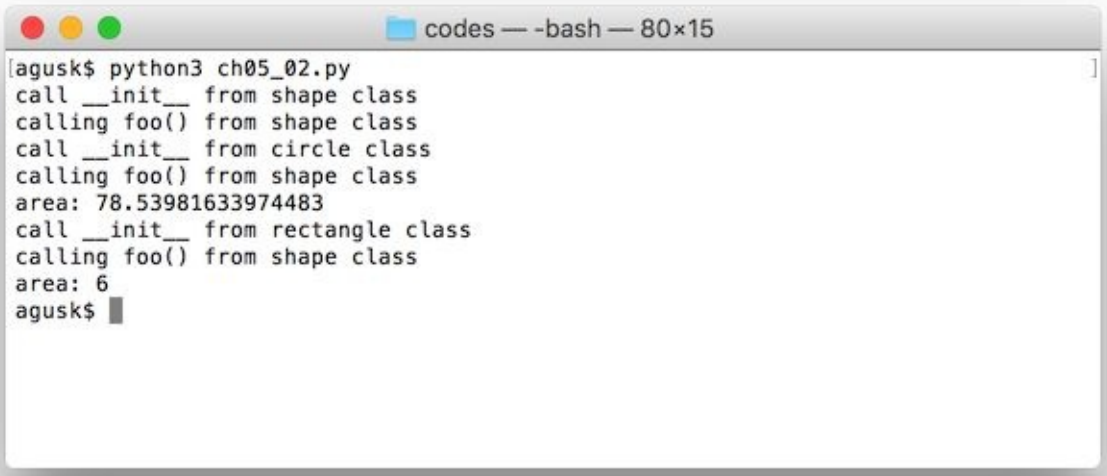
```
import math


class shape:
    def __init__(self):
        print('call __init__ from shape class')

    def foo(self):
        print('calling foo() from shape class')


class circle(shape):
    def __init__(self, r):
        print('call __init__ from circle class')
        self.r = r

    def calculate_area_circle(self):
        return math.pi * self.r * self.r


class rectangle(shape):
    def __init__(self, l, w):
        print('call __init__ from rectangle class')
        self.l = l
        self.w = w

    def calculate_area_rectangle(self):
        return self.l * self.w


a = shape()
a.foo()

b = circle(5)
b.foo()
area = b.calculate_area_circle()
print('area:', area)

c = rectangle(2, 3)
c.foo()
area = c.calculate_area_rectangle()
print('area:', area)
```

Save these scripts into a file, called ch05_02.py.

Now run this file.

```
$ python3 ch05_02.py
```

Program output:



5.7 Overriding Methods

We can override class methods in OOP. For instance, we have a class, shape, which has calculate_area(). We override this method from derived class by our own implementation.

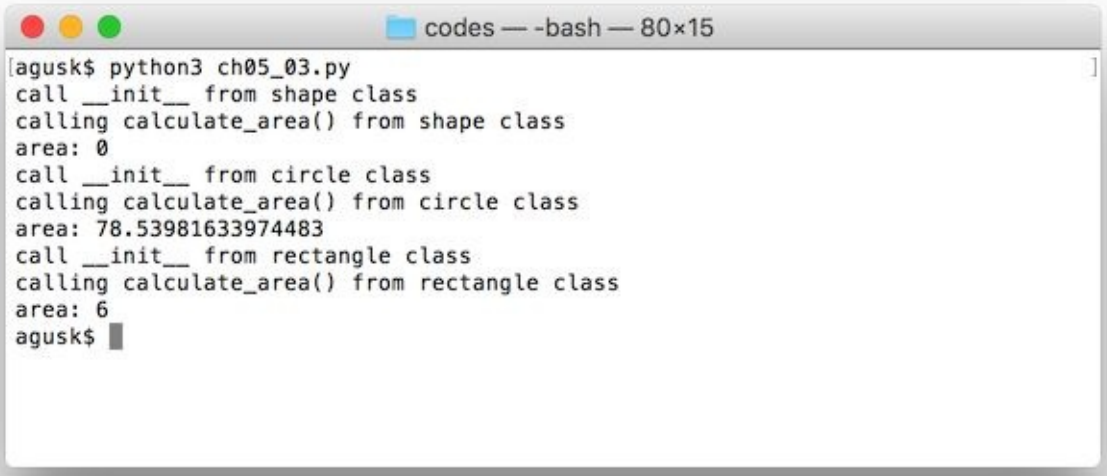
Write these scripts for sample.

```
import math


class shape:
    def __init__(self):
        print('call __init__ from shape class')

    def calculate_area(self):
        print('calling calculate_area() from shape class')
        return 0


class circle(shape):
    def __init__(self, r):
        print('call __init__ from circle class')
        self.r = r

    def calculate_area(self):
        print('calling calculate_area() from circle class')
        return math.pi * self.r * self.r


class rectangle(shape):
    def __init__(self, l, w):
        print('call __init__ from rectangle class')
        self.l = l
        self.w = w

    def calculate_area(self):
        print('calling calculate_area() from rectangle class')
        return self.l * self.w


a = shape()
area = a.calculate_area()
print('area:', area)

b = circle(5)
area = b.calculate_area()
print('area:', area)

c = rectangle(2, 3)
area = c.calculate_area()
print('area:', area)
```

Save and run the program.

```
$ python3 ch05_03.py
```

A sample of program output:



5.8 Overloading Operators

We can define our overloading operators in Python using __add__() for instance.

For completed, write these scripts.

```
class Point:

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __add__(self, other):
        return Point(self.x + other.x, self.y + other.y)

    def __sub__(self, other):
        return Point(self.x - other.x, self.y - other.y)

    def __mul__(self, other):
        return Point(self.x * other.x, self.y * other.y)

    def __str__(self):
        return 'x' + str(self.x) + ', y:' + str(self.y)


a = Point(10, 3)
b = Point(2, 7)
c = Point(8, 1)

print(a)
print(a + b)
print(c - b)
print(a * c)
```

Save these scripts into a file, called ch05_04.py.

```
$ python3 ch05_04.py
```

A program output:

codes — -bash — 80×15
agusk$ python3 ch05_04.py
x10, y:3
x12, y:10
x6, y:-6
x80, y:3
agusk$

6. Python Modules and Packages

This chapter explains how to work Python modules and packages.

6.1 Python Modules

In this chapter, we learn how to access Python modules. A list of Python package can be found this website, https://pypi.python.org/pypi?%3Aaction=index .

6.2 import Statement

You can access Python module using import. Try to write these scripts.

```
import math


a = math.sin(0.3)
print(a)
b = math.sqrt(math.sin(0.5) * math.pow(5, 3))
print(b)
```

Save into a file, called ch06_01.py. Run the program.

```
$ python3 ch06_01.py
```

Program output:

6.3 from…import * Statement

On previous section, we use import to use Python module. To use it, you should module name. You can ignore it using from…import statement.

Write these scripts.

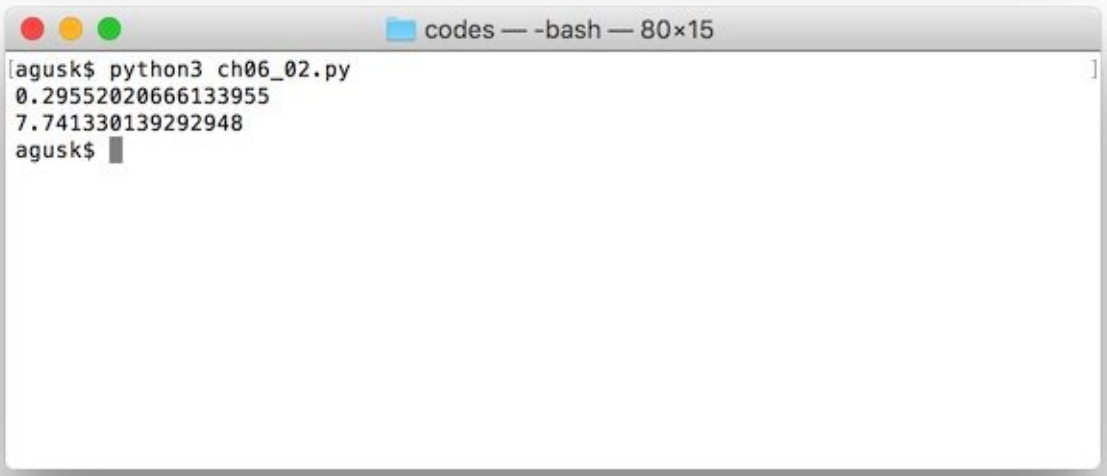
```
from math import *


a = sin(0.3)
print(a)
b = sqrt(sin(0.5) * pow(5, 3))
print(b)
```

You can see we don’t need to call math.sin(). We just call sin().

Save and ryn the program.

```
$ python3 ch06_02.py
```

Program output:



6.4 Installing External Python Package

If you want to use external Python Package, for instance,
Colorama, https://pypi.python.org/pypi/colorama . We can install itu via pip.

Type this command on Terminal.

```
pip install colorama
```

If you want to install it for Python 3.x, you should pip3.

```
pip3 install colorama
```

Now let's write the demo.

```
import colorama
from colorama import Fore, Back, Style

colorama.init()
message = "hello world from python"

print(message)
print(Fore.RED + message)
print(Fore.GREEN + message)
print(Fore.BLUE + message)
print(Fore.RED + Back.YELLOW + message + Style.RESET_ALL)
```

Save into a file, called ch06_03.py.

```
$ python3 ch06_03.py
```

Program output:

codes — -bash — 80×15
agusk$ python3 ch06_03.py
hello world from python
hello world from python
hello world from python
hello world from python
hello world from python
agusk$

7. String Operations

This chapter explains how to work with String operation in Python.

7.1 Getting Started

We already use string as data type, https://docs.python.org/3/library/string.html . In this section, we explore some operations in string.

The next step is to explore how to work with string.

7.2 Concatenating Strings

If you have a list of string, you can concatenate into one string. You can use + operator and format() function. Here is a sample code

```
# Concatenating
print(str1 + " " + str2)
print(str1, str2)
print("%s %s" % (str1, str2))
print("{} {}".format(str1, str2))
```

7.3 String To Numeric

Sometime you want to do math operations but input data has string type. To convert string type into numeric, you can use int() for String to Integer and float() for string to Float.

The following is a sample code to implement string to numeric conversion.

```
# string to numeric
a = "2"
b = "6.8"

num1 = int(a)
num2 = float(b)
print(num1)
print(num2)
```

7.4 Numeric to String

It is easy to convert numeric to String type, you can use str(). You can get string type automatically.

```
# numeric to string
a = 6
b = 8.56

str1 = str(a)
str2 = str(b)
print(str1)
print(str2)
```

7.5 String Parser

The simple solution to parsing String uses *split()* with delimiter parameter. For example, you have String data with ; delimiter and want to parse it. Here is sample code

```
# parsing
msg = 'Berlin;Amsterdam;London;Tokyo'
cities = msg.split(';')
for city in cities:
    print(city)
```

7.6 Check String Data Length

You can use *len()* to get the length of data.

```
msg = 'Hello world, Python!'

# get a length of string
length = len(msg)
print(length)
```

7.7 Copy Data

You may copy some characters from String data. To do it, you can use [start:end] syntax. Here is syntax format:

```
msg = 'Hello world, Python!'

# copy
print(msg[5:])
print(msg[:5])
print(msg[-3:])
print(msg[:-3])
print(msg[2:6])
print(msg[5:8])
```

7.8 Upper and Lower Case Characters

In some situation, you want to get all string data in upper or lower case characters. This feature is built in String object. *upper()* function is used to make whole string in upper case and *lower()* is used to make whole string in lower case.

The following is a sample code to get upper and lower case characters.

```
msg = 'Hello world, Python!'

# upper & lower
print(msg.upper())
print(msg.lower())
```

7.9 Testing A Program

We can write our code in ch17_01.py completely as follows.

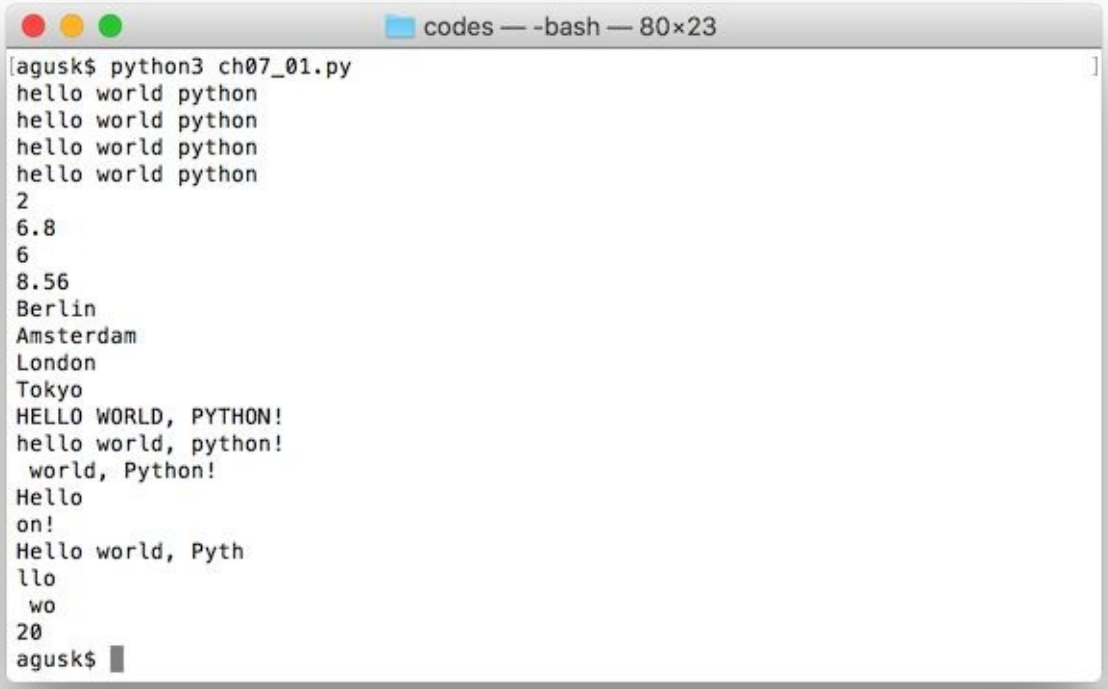
```
str1 = "hello world"
str2 = "python"

# Concatenating
print(str1 + " " + str2)
print(str1, str2)
print("%s %s" % (str1, str2))
print("{} {}".format(str1, str2))


# string to numeric
a = "2"
b = "6.8"

num1 = int(a)
num2 = float(b)
print(num1)
print(num2)


# numeric to string
a = 6
b = 8.56

str1 = str(a)
str2 = str(b)
print(str1)
print(str2)


# parsing
msg = 'Berlin;Amsterdam;London;Tokyo'
cities = msg.split(';')
for city in cities:
    print(city)


# string operations
msg = 'Hello world, Python!'

# upper & lower
print(msg.upper())
print(msg.lower())

# copy
print(msg[5:])
print(msg[:5])
print(msg[-3:])
```

```
print(msg[:-3])
print(msg[2:6])
print(msg[5:8])

# get a length of string
length = len(msg)
print(length)
```

Save the script and run the program.

```
$ python3 ch07_01.py
```

Program output:



8. File Operations

This chapter explains how to work with file operations using Python.

8.1 Getting Started

We can work with I/O file using io package, https://docs.python.org/3/library/io.html .

The next step is to build Python application to write and read a file.

8.2 Writing Data Into A File

To write and read a file, we can use io package. In this section, we try to write data into a file.

8.2.1 Creating a File

We can create a file using open() function with parameter “w”. If file is exist, it will recreate a file.

If you want to use the existing file, you can pass “a”. Parameter “b” is used for binary file.

```
# create a file.
# If file is existing, it erases and creates a new one
f1 = open('mydoc1', 'w')


# create a file.
# If file is existing, it appends. Otherwise, it creates
f2 = open('mydoc2', 'a')


# binary files
bf1 = open('mydoc3', 'wb')
bf2 = open('mydoc4', 'ab')
```

8.2.2 Writing Data

Write data into a file, we can use write() function.

```
for index in range(1, 12):
    data = ''
    name = 'user ' + str(index-1)
    email = 'user' + str(index-1) + '@email.com'
    if index == 1:
        data = '{0:3s} {1:10s} {2:15s}\n'.format('No', 'Name', 'Email')
    else:
        data = '{0:3s} {1:10s} {2:15s}\n'.format(str(index-1), name, ema

    f1.write(data)
    f2.write(data)
    bf1.write(data)
    bf2.write(data)
```

8.2.3 Closing a File

If file operations done, you should call close() to close file.

```
f1.close()
f2.close()
bf1.close()
bf2.close()
```

8.2.4 Demo

Let’s write these scripts for demo.

```
#######################################
print('creating files…')

# create a file.
# If file is existing, it erases and creates a new one
f1 = open('mydoc1', 'w')


# create a file.
# If file is existing, it appends. Otherwise, it creates
f2 = open('mydoc2', 'a')


# binary files
bf1 = open('mydoc3', 'wb')
bf2 = open('mydoc4', 'ab')


#######################################
# writing data

print('writing data into files…')
for index in range(1, 12):
    data = ''
    name = 'user ' + str(index-1)
    email = 'user' + str(index-1) + '@email.com'
    if index == 1:
        data = '{0:3s} {1:10s} {2:15s}\n'.format('No', 'Name', 'Email')
    else:
        data = '{0:3s} {1:10s} {2:15s}\n'.format(str(index-1), name, ema

    f1.write(data)
    f2.write(data)
    bf1.write(data)
    bf2.write(data)
```

```
#####################################
# close all

print('close files…')
f1.close()
f2.close()
bf1.close()
bf2.close()
```

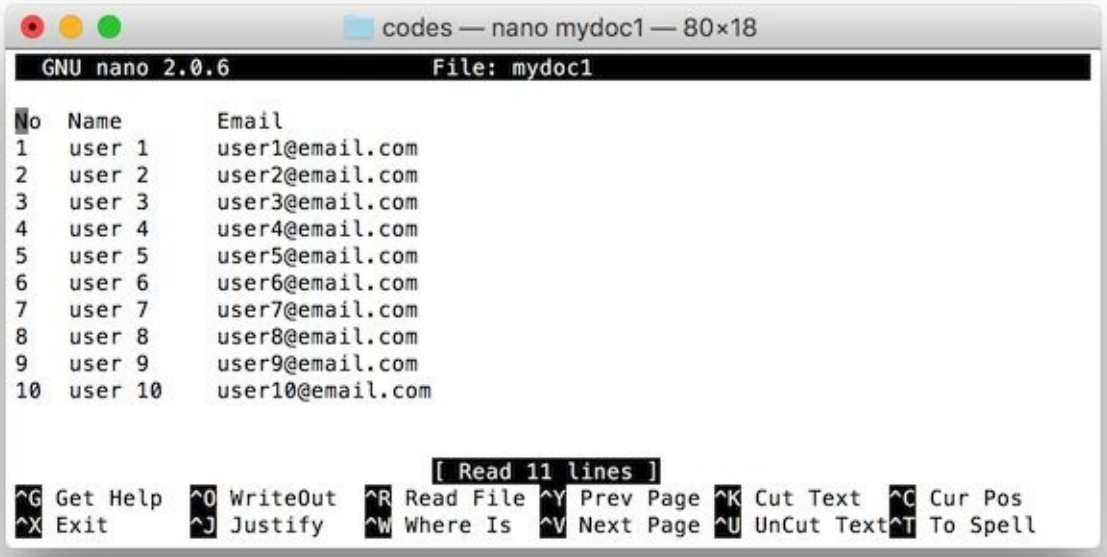
Save into a file, called ch08_01.py. Then, run the program.

```
$ python3 ch08_01.py
```

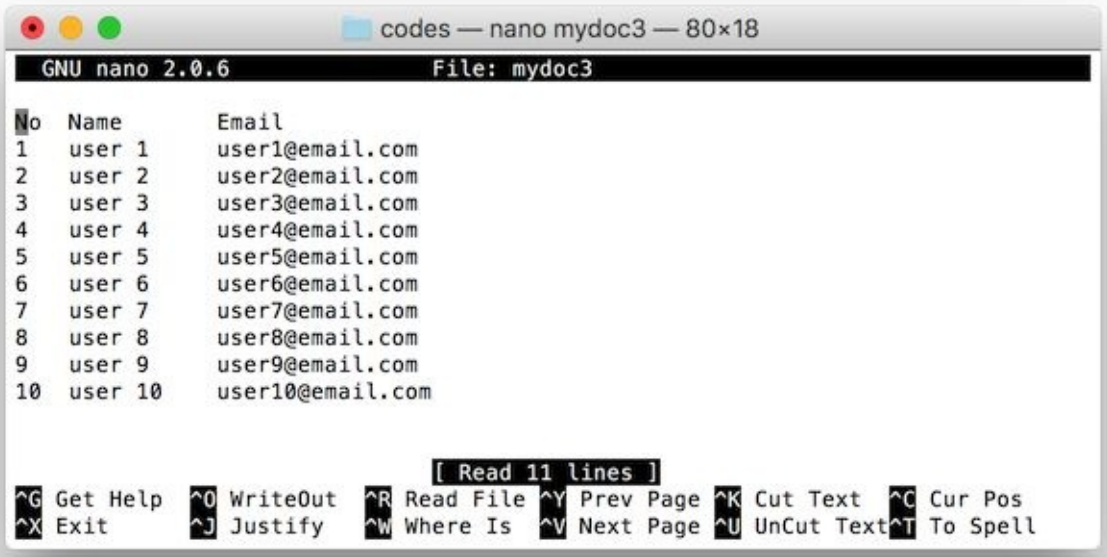
Program output:

If success, you can open all files to see the content.

A sample of content from mydoc1 file can be seen in Figure below.



A sample of content from mydoc3 file can be seen in Figure below.



8.3 Reading Data From A File

To read data per line from a file, we use readline() function.

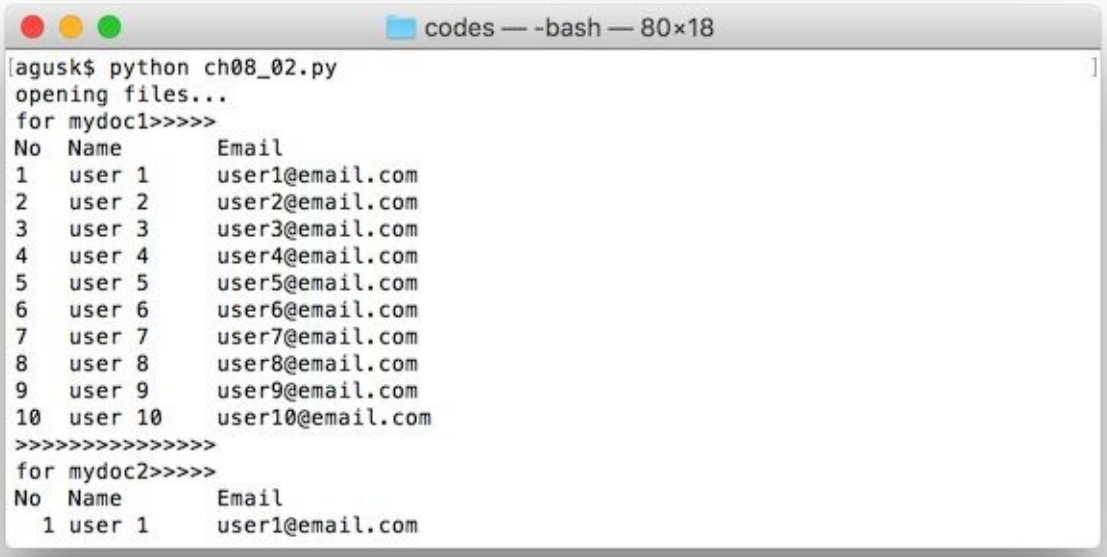
Write these scripts for demo.

```
import sys

#######################################
print('opening files…')

f1 = open('mydoc1', 'r')
f2 = open('mydoc2', 'r')
bf1 = open('mydoc3', 'rb')
bf2 = open('mydoc4', 'rb')


#######################################
# reading data

def reading_data(f):
    while True:
        data = f.readline()
        if (data == '') or (data == None):
            break

        sys.stdout.write(data)


print('for mydoc1>>>>>')
reading_data(f1)
print('>>>>>>>>>>>>>>>>')

print('for mydoc2>>>>>')
reading_data(f2)
print('>>>>>>>>>>>>>>>>')

print('for mydoc3>>>>>')
reading_data(bf1)
print('>>>>>>>>>>>>>>>>')

print('for mydoc4>>>>>')
reading_data(bf1)
print('>>>>>>>>>>>>>>>>')

#######################################
# close all

print('close files…')
f1.close()
f2.close()
bf1.close()
bf2.close()
```

Save into a file, called ch08_02.py. Then, run the program.

```
$ python3 ch08_02.py
```

Program output:



9. Error Handling

This chapter explains how to handle errors and exceptions that occur in Python application.

9.1 Error Handling

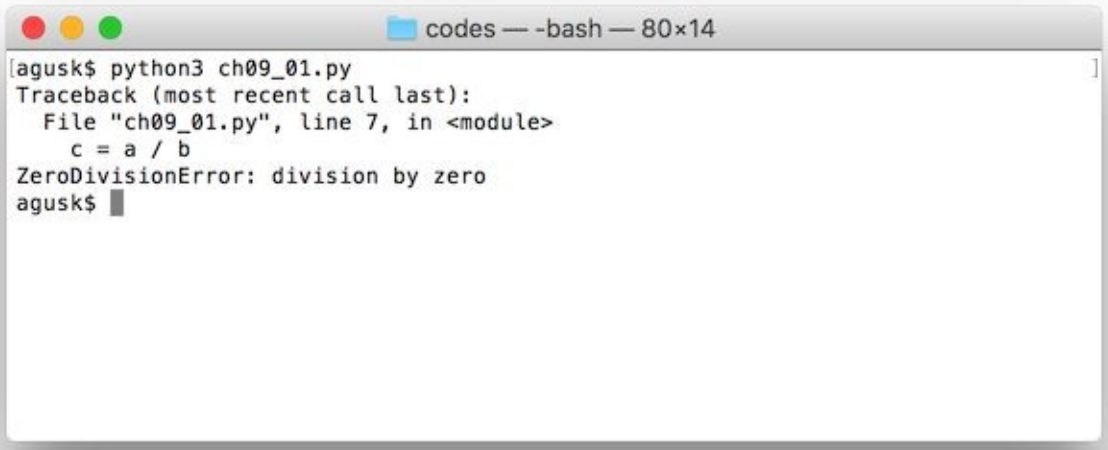
Basically when we write a program and occur error on running, Python will catch program error.

Try to write these scripts and run this program.

```
a = 18
b = 0

c = a / b
```

```
$ python3 ch09_01.py
```

You should get error, shown in Figure below.



Now we can catch error using try..except. You can read how to use it on https://docs.python.org/3/tutorial/errors.html .

Write these scripts.

```
try:
    a = 18
    b = 0

    c = a / b
    print('result:', str(c))

except ZeroDivisionError as e:
    print('Error: division by zero')
    print(e)

finally:
```

```
    print('Done')

print('exit from program')
```

Save into a file, ch09_02.py. Then, run the file.

```
$ python3 ch09_02.py
```

You can see the program can handle the error.

9.2 Catching All Errors

On previous section, we catch error for “division by error”. We can catch all errors using Exception object. Write these scripts for demo.

```
try:
    a = 18
    b = 0

    c = a / b
    print('result:', str(c))

except Exception as e:
    print(e)

finally:
    print('Done')

print('exit from program')
```

Save into a file, called ch09_03.py. Run the program.

```
$ python3 ch09_03.py
```

Program output:

9.3 Raising Exceptions

We can raise error from our program using raise.

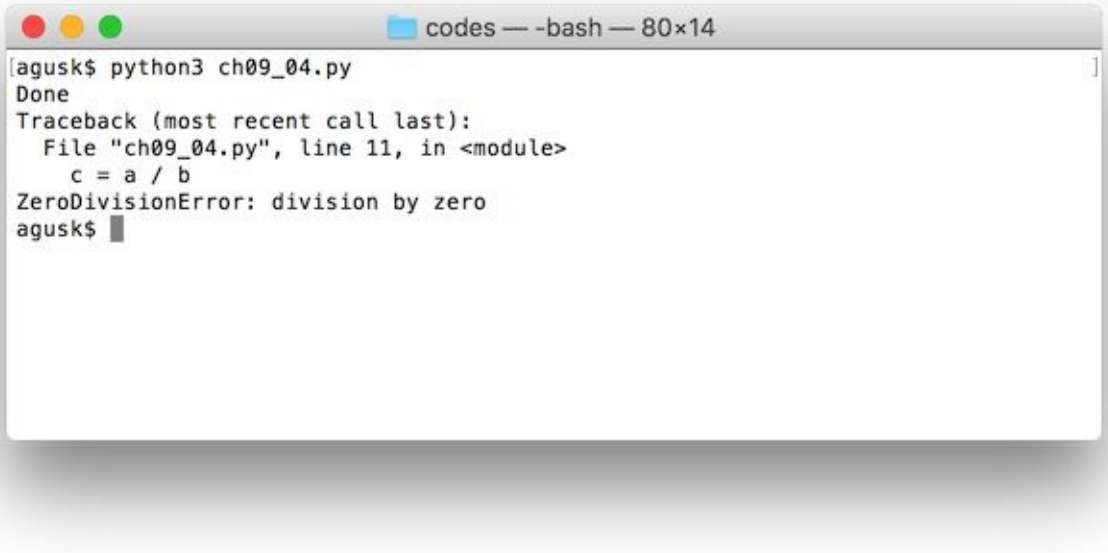
For demo, write these scripts.

```
try:
    a = 18
    b = 0

    c = a / b
    print('result:', str(c))

except Exception as e:
    raise

finally:
    print('Done')

# this code is never called
print('exit from program')
```

Save into a file, called ch09_04.py. Then, run the program.

```
$ python3 ch09_04.py
```

You should the program raise the error so we don't words "exit from program".



9.4 Custom Exception

We can build own error with implementing inheritance Exception.

For instance, we create a class, MySimpleError, with inheritance from Exception.

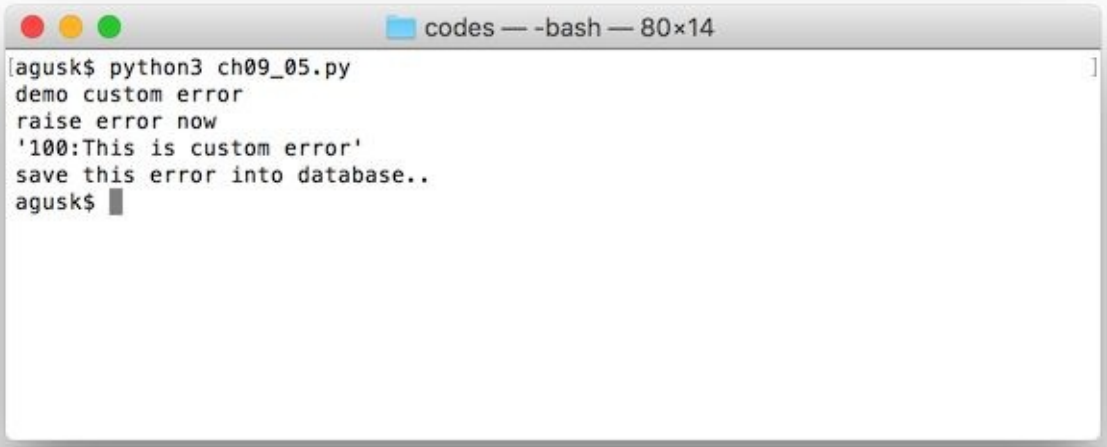
```
class MySimpleError(Exception):
    def __init__(self, code, message):
        self.code = code
        self.message = message

    def __str__(self):
        return repr(str(self.code) + ":" + self.message)

    def save_to_database(self):
        print('save this error into database..')


# how to use custom error
try:
    print('demo custom error')
    print('raise error now')
    raise MySimpleError(100,'This is custom error')

except MySimpleError as e:
    print(e)
    e.save_to_database()
```

Save the program into a file, called ch09_05.py.

Run this program.

```
$ python3 ch09_05.py
```

Program output:


codes — -bash — 80×14
agusk$ python3 ch09_05.py
demo custom error
raise error now
'100:This is custom error'
save this error into database..
agusk$

10. Building Own Python Module

This chapter explains how to build own Python module.

10.1 Creating Simple Module

In this section, we create a simple module. We will call functions from external file (*.py) in the same package, main package.

Firstly, create a file, simplemodule.py, and write these scripts.

```
def perform(a, b):
    return a * 2.5 + b


def calculate(a,b):
    return a + b * 5
```

You also can create a class on Python file, for instance simpleadvmodule.py and write these scripts.

```
from math import *


class City:
    # class data
    city_count = 0
    city_id = 0

    # constructor
    def __init__(self, name='', x=0, y=0):
        self.name = name
        self.x = x
        self.y = y
        City.city_count += 1  # access all City classes
        self.city_id = City.city_count

    def __str__(self):
        return 'City: ' + self.name + ',id=' + str(self.city_id) + ',x='

    # class attributes
    def move_to(self, x=0, y=0):
        self.x += x
        self.y += y

    def distance(self, other_city):
        xi = pow(other_city.x - self.x, 2)
        yi = pow(other_city.y - self.y, 2)

        return sqrt(xi + yi)

    def __del__(self):
        # get class name
        class_name = self.__class__.__name__
        print('class ', class_name, ' destroyed')
```

Now you access functions from simplemodule.py and simpleadvmodule.py files using import statement.

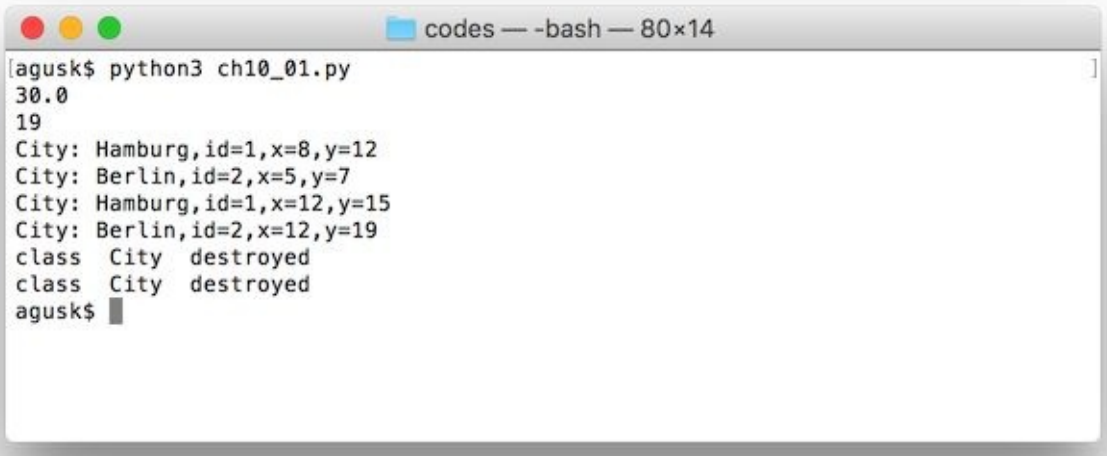
Write these scripts.

```
# access our modules
import simplemodule
import simpleadvmodule

# use simplemodule
num1 = simplemodule.perform(10, 5)
print(num1)

num1 = simplemodule.calculate(4, 3)
print(num1)

# use simpleadvmodule
city_a = simpleadvmodule.City('Hamburg', 8, 12)
city_b = simpleadvmodule.City('Berlin', 5, 7)
print(city_a)
print(city_b)

city_a.move_to(4, 3)
city_b.move_to(7, 12)
print(city_a)
print(city_b)
```

Save into a file, called ch10_01.py. Run the program.

```
$ python3 ch10_01.py
```

Program output:

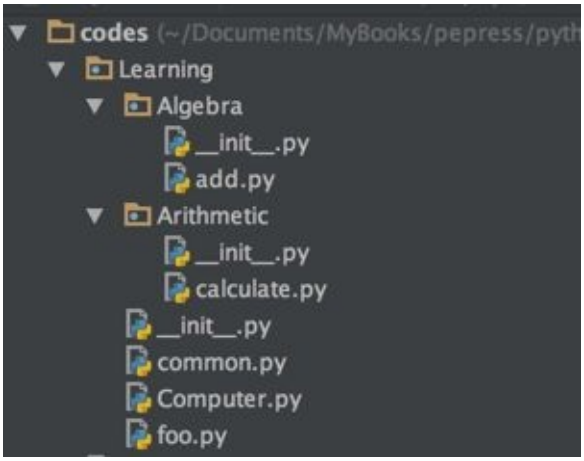


10.2 Building Own Python Package

In previous program, we create a module in Python package. We create a package and then use it in our program.

You can create a package, called Learning, by creating folder. Inside Learning folder, you create folders: Algebra and Arithmetic.

The following is our package structure.



Now we add several files in each folder

- Learning folder: __init__.py, common.py, Computer.py and foo.py
- Learning/Algebra folder: __init__.py and add.py
- Learning/Arithmetic folder: __init__.py and calculate.py

The following is script implementation for each file.

common.py

```
def do_something():
    print('call do_something()')
```

Computer.py

```
class Computer:

    # constructor
    def __init__(self, name=''):
        self.name = name

    def __str__(self):
        return 'Computer: ' + self.name

    def say_hello(self):
        print("I'm computer, called", self.name)
```

foo.py

```
def foo():
    print('call foo()')
```

__init__.py from Learning folder.

```
from common import do_something
from Computer import Computer
from foo import foo
```

add.py

```
def add(a, b):
    return a + b
```

__init__.py from Learning/Algebra folder.

```
from Learning.Algebra.add import add
```

calculate.py

```
def calculate(a, b):
    return a + b * 2.8
```

__init__.py from Learning/Arithmetic folder.

```
from Learning.Arithmetic.calculate import calculate
```

Now we can access our package. Create a file, called ch10_02.py, and write these scripts.

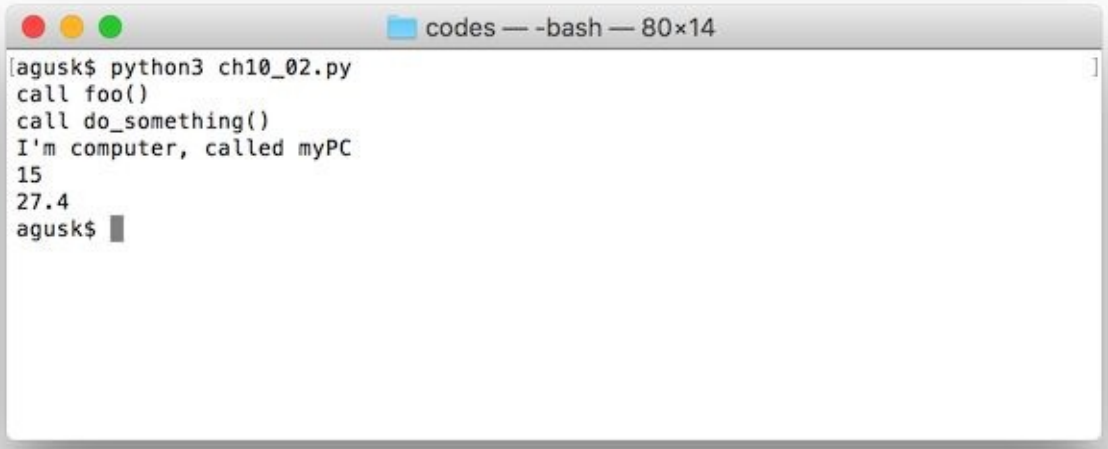
```
import sys
sys.path.append('./Learning')

import Learning

Learning.foo()
Learning.do_something()
a = Learning.Computer('myPC')
a.say_hello()


import Learning.Algebra as algebra
b = algebra.add(10, 5)
print(b)
```

```
import Learning.Arithmetic as arith
c = arith.calculate(5, 8)
print(c)
```

Save and run the program.

```
$ python3 ch10_02.py
```

Program output:



11. Concurrency

This chapter explains how to create concurrency in Python

11.1 Getting Started

We can run a program or a function in background using Python. In this chapter, we explore several scenarios to build concurrency application. In Python, you can read concurrency on this website, https://docs.python.org/3/library/concurrency.html .

11.2 Threading

Basically, we can implement threading using Thread with passing the function. You can call start() to run a thread.

```
import time
import threading

global running


def perform():
    global running

    counter = 0
    running = True
    while running:
        print('counter:', str(counter))
        time.sleep(2)
        counter += 1


my_thread = threading.Thread(target=perform)
my_thread.setDaemon(True)
my_thread.start()

# python 3
input("Press Enter to stop…")

# python 2
#raw_input("Press Enter to stop…")

running = False
my_thread.join(2)
```

To exit from a thread, we can call join() with timeout value.

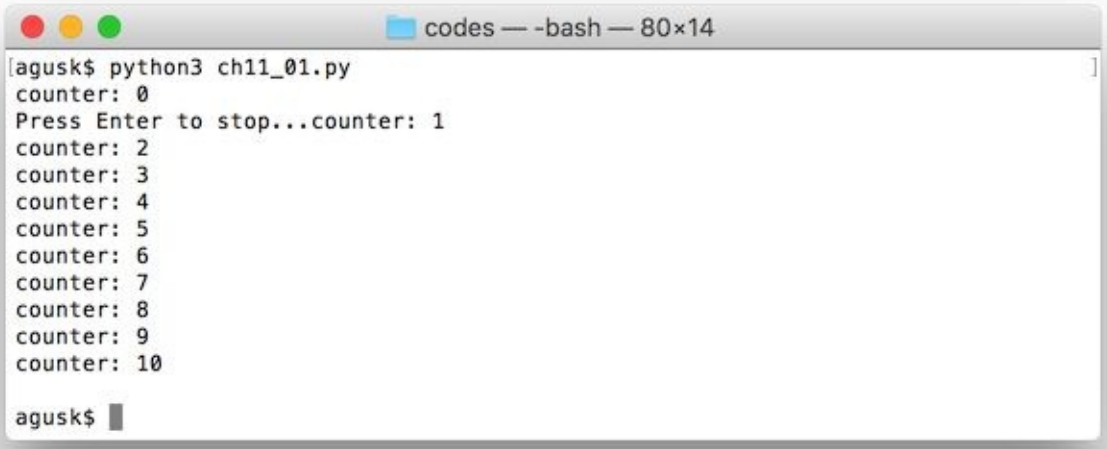
Save into a file, called ch11_01.py.

Run the program.

```
$ python3 ch11_01.py
```

When Thread.start() is called, it executes perform() function.

A sample of program output can be seen in Figure below.



Thread object can be implemented by a derived class from threading.Thread. For instance, we create a class, MyThread. It has inheritance from threading.Thread and implement run() function.

Write these scripts.

```
import time
import threading


class MyThread(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        self.running = False

    def run(self):
        counter = 0
        self.running = True
        while self.running:
            print('counter:', str(counter))
            time.sleep(2)
            counter += 1

    def stop(self):
        print('stopping thread…')
        self.running = False
        self.join(2)


my_thread = MyThread()
my_thread.setDaemon(True)
my_thread.start()

# python 3
```

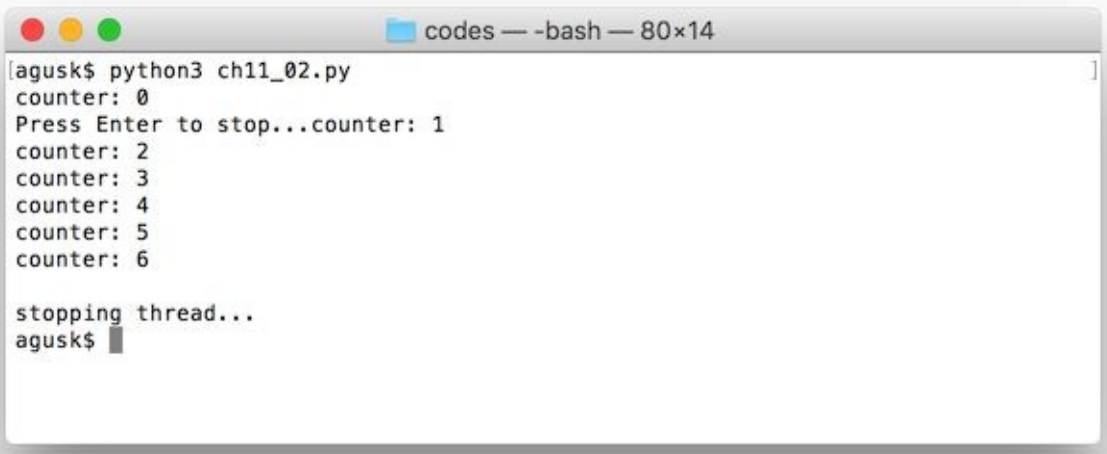
```
input("Press Enter to stop…")

# python 2
#raw_input("Press Enter to stop…")

my_thread.stop()
```

Save these scripts into a file, called ch11_02.py. You can see our object, my_thread, call start() then, it call run(). We call stop() function to stop our thread. Basically, MyThread will call join() while called stop() function.

Now you can run the program.

```
$ python3 ch11_02.py
```

A sample of program output:



11.3 Synchronizing Threads

We can synchronize among background codes in Python. In this section, we use mutex lock and event for thread synchronization.

Let’s start.

11.3.1 Mutex Locks

The idea is a simple. When we access a resource, we call acquire(). If done, you call release() from Lock object.

For testing, we define a shared resource, such as a variable called value. This variable will be access by two threads. Only one thread can access this variable.

Ok, write these scripts.

```
import time
import threading


class MyThread(threading.Thread):
    def __init__(self, name, o_lock):
        threading.Thread.__init__(self)
        self.name = name
        self.running = False
        self.value_lock = o_lock

    def run(self):
        global value
        self.running = True
        while self.running:
            self.value_lock.acquire()
            value += 1
            print('value:', str(value),' from ', self.name)
            self.value_lock.release()
            time.sleep(2)

    def stop(self):
        print('stopping ', self.name)
        self.running = False
        self.join(2)


global value
value = 0
value_lock = threading.Lock()

my_thread1 = MyThread('Thread 1',value_lock)
```

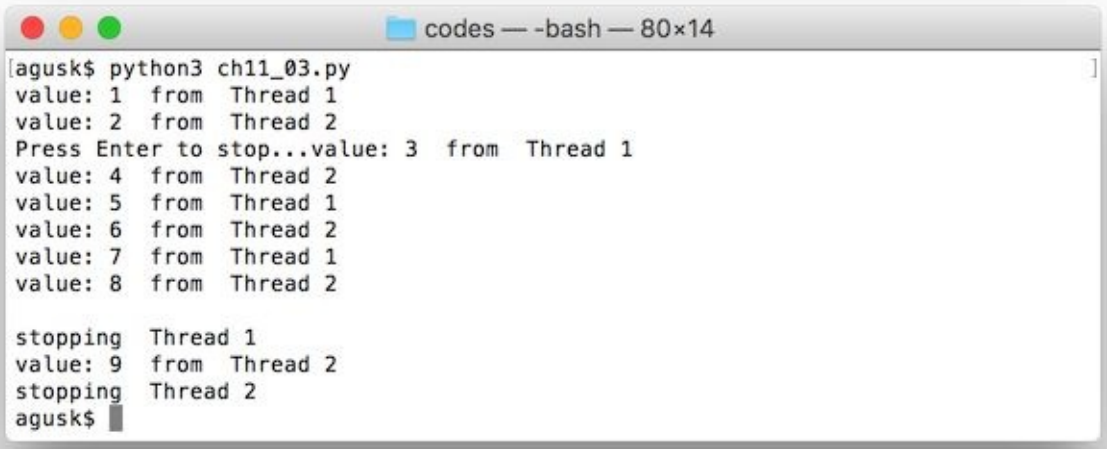
```
my_thread1.setDaemon(True)
my_thread2 = MyThread('Thread 2',value_lock)
my_thread2.setDaemon(True)

my_thread1.start()
my_thread2.start()
# python 3
input("Press Enter to stop…")

# python 2
#raw_input("Press Enter to stop…")

my_thread1.stop()
my_thread2.stop()
```

Save into a file, called ch11_03.py.

Now you can run the program.

```
$ python3 ch11_03.py
```

Program output:



11.3.2 Event

Another option to synch threading, we can use Event object. Call wait() to block operation. It means the program can’t execute codes after wait(). Then, call set() to release the blocking process.

For illustration, we create three worker threads. These threads will perform something

after calling set() from event. This is useful for initialization state process.

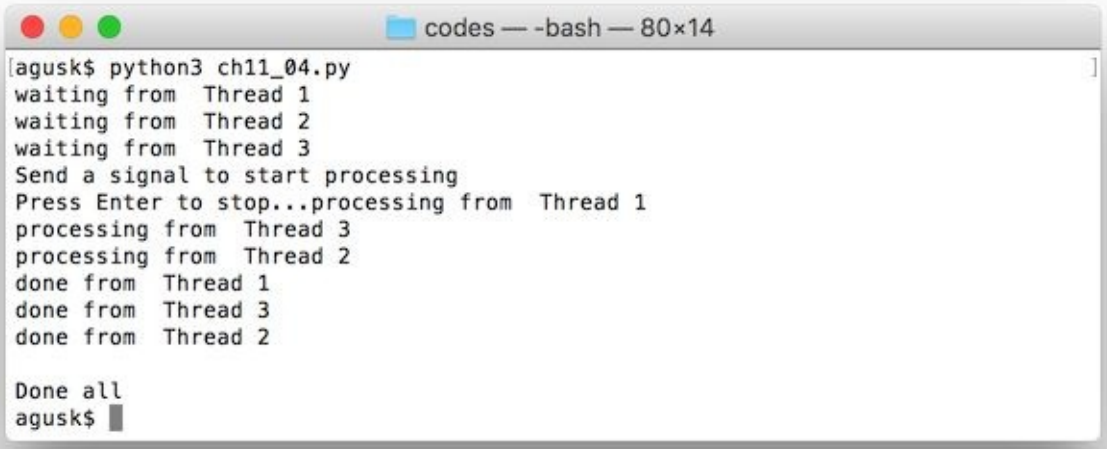
Write these scripts.

```
import time
import threading


class Worker(threading.Thread):
    def __init__(self, name, signal):
        threading.Thread.__init__(self)
        self.name = name
        self.signal = signal

    def run(self):
        print('waiting from ', self.name)
        self.signal.wait()
        print('processing from ', self.name)
        time.sleep(2)
        print('done from ', self.name)


signal_event = threading.Event()
my_thread1 = Worker('Thread 1', signal_event)
my_thread1.setDaemon(True)
my_thread2 = Worker('Thread 2', signal_event)
my_thread2.setDaemon(True)
my_thread3 = Worker('Thread 3', signal_event)
my_thread3.setDaemon(True)

my_thread1.start()
my_thread2.start()
my_thread3.start()

# waiting for 10 seconds
time.sleep(10)

# start process
print('Send a signal to start processing')
signal_event.set()

# python 3
input("Press Enter to stop…")

# python 2
#raw_input("Press Enter to stop…")

print('Done all')
```

Save into a file, called ch11_04.py. Then, run the program.

```
$ python3 ch11_04.py
```

Program output:



11.4 Queue

In this section, we learn about Queue object from Python, https://docs.python.org/3/library/queue.html .

For testing, we add some jobs into Queue. Then, some worker threads will peak the job and run it.

Let's write these scripts.

```
import time
import threading
import queue


class Worker(threading.Thread):
    def __init__(self, name, q):
        threading.Thread.__init__(self)
        self.name = name
        self.q = q

    def run(self):
        while True:
            if self.q.empty():
                print('thread stopped')
                break
            job = self.q.get()
            print('run job', str(job), ' from', self.name)
            time.sleep(1)
            self.q.task_done()


q = queue.Queue()
# generate jobs
print('populate jobs')
for i in range(15):
    q.put(i)


my_thread1 = Worker('Thread 1', q)
my_thread1.setDaemon(True)
my_thread2 = Worker('Thread 2', q)
my_thread2.setDaemon(True)
my_thread3 = Worker('Thread 3', q)
my_thread3.setDaemon(True)

my_thread1.start()
my_thread2.start()
my_thread3.start()

my_thread1.join()
my_thread2.join()
my_thread3.join()
```

```
# python 3
input("Press Enter to stop…")

# python 2
#raw_input("Press Enter to stop…")

print('Done all')
```

Save into a file, called ch11_05.py.

Now you can run the program.

```
$ python3 ch11_05.py
```

Program output:

11.5 Multiprocessing

We can implement concurrency in Python using multiprocessing. You can read the information about this on this site, https://docs.python.org/3/library/multiprocessing.html .

11.5.1 Process

We can use Process object to implement multiprocessing in Python. For instance, we build a counter from a process.

Write these scripts.

```
import time
import multiprocessing


class MyProcess(multiprocessing.Process):
    def __init__(self):
        multiprocessing. Process.__init__(self)
        self.running = False

    def run(self):
        counter = 0
        self.running = True
        while self.running:
            print('counter:', str(counter))
            time.sleep(2)
            counter += 1

    def stop(self):
        print('stopping process…')
        self.running = False
        self.join(1)


my_process = MyProcess()
my_process.daemon = True
my_process.start()

# python 3
input("Press Enter to stop…")

# python 2
#raw_input("Press Enter to stop…")

my_process.stop()
```

Save into a file, called ch11_06.py, and run it.

```
$ python3 ch11_06.py
```

Program output:

11.5.2 Synchronizing Processes

We can synch among processes using multiprocessing.Value. This object implement synchronizing process.

For testing, we define a shared resource via multiprocessing.Value. Write these scripts.

```
import time
import multiprocessing


class MyProcess(multiprocessing.Process):
    def __init__(self, name, shared_dt):
        multiprocessing. Process.__init__(self)
        self.name = name
        self.running = False
        self.shared_data = shared_dt

    def run(self):
        self.running = True
        while self.running:
            time.sleep(1)
            with self.shared_data.get_lock():
                self.shared_data.value += 1
                print('value:', str(self.shared_data.value), ' from ', s
```

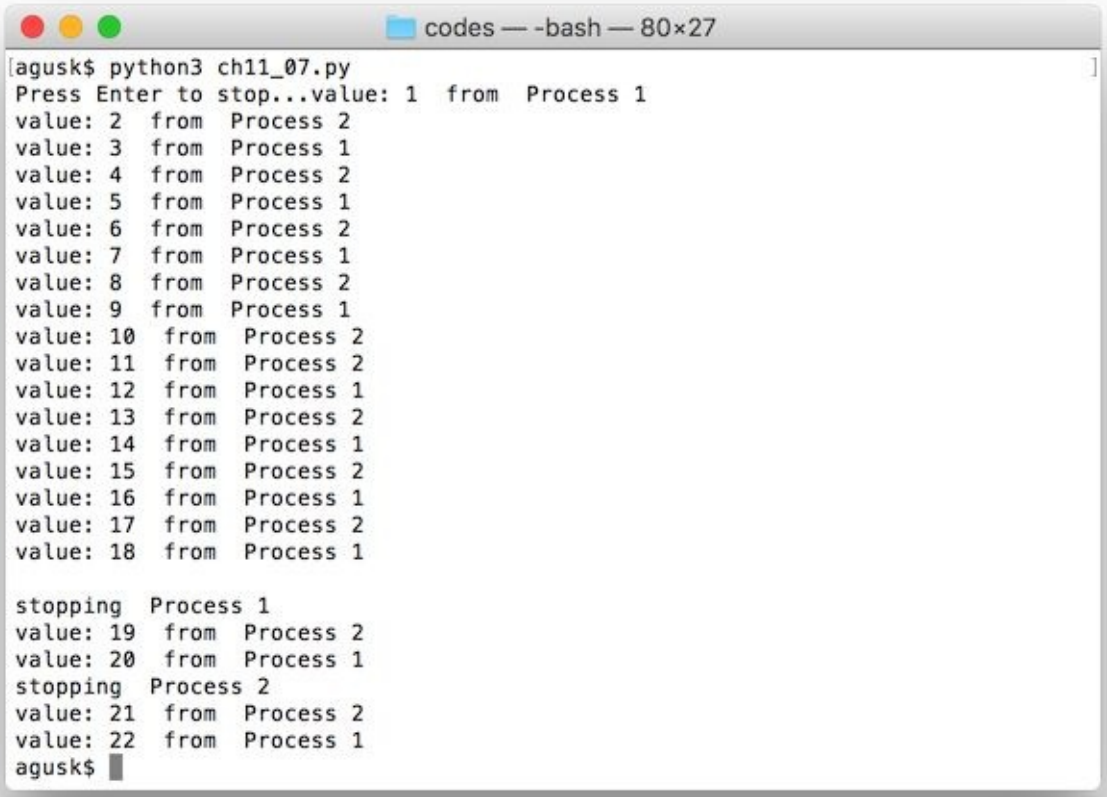
```
    def stop(self):
        print('stopping ', self.name)
        self.running = False
        self.join(1)


shared_data = multiprocessing.Value('i', 0, lock=True)

my_process1 = MyProcess('Process 1', shared_data)
my_process1.daemon = True
my_process2 = MyProcess('Process 2', shared_data)
my_process2.daemon = True

my_process1.start()
my_process2.start()

# python 3
input("Press Enter to stop…")

# python 2
#raw_input("Press Enter to stop…")

my_process1.stop()
my_process2.stop()
```

Save the program into a file, called ch11_07.py.

Now you can run the program.

```
$ python3 ch11_07.py
```

Program output:


codes — -bash — 80×27
agusk$ python3 ch11_07.py
Press Enter to stop...value: 1 from Process 1
value: 2 from Process 2
value: 3 from Process 1
value: 4 from Process 2
value: 5 from Process 1
value: 6 from Process 2
value: 7 from Process 1
value: 8 from Process 2
value: 9 from Process 1
value: 10 from Process 2
value: 11 from Process 2
value: 12 from Process 1
value: 13 from Process 2
value: 14 from Process 1
value: 15 from Process 2
value: 16 from Process 1
value: 17 from Process 2
value: 18 from Process 1
stopping Process 1
value: 19 from Process 2
value: 20 from Process 1
stopping Process 2
value: 21 from Process 2
value: 22 from Process 1
agusk$

11.6 Parallel Tasks

The last section is to implement parallel tasks using concurrent.futures. There are two options to implement this: ThreadPoolExecutor and ProcessPoolExecutor.

11.6.1 ThreadPoolExecutor

ThreadPoolExecutor uses thread to do parallel tasks.

A sample of script for parallel tasks can be written the following script.

```
import queue
import concurrent.futures
import random
import time
import datetime


def perform(q, a, b, c):
    rand_val = random.uniform(0, 2)
    res = a * b * 10 - c * 2
    time.sleep(rand_val)

    q.put(res)


t1 = datetime.datetime.now()
q = queue.Queue()
with concurrent.futures.ThreadPoolExecutor(max_workers=3) as executor:
    for i in range(1, 15):
        val_a = random.randint(1, 10)
        val_b = random.randint(1, 10)
        val_c = random.randint(1, 10)
        executor.submit(perform, q, val_a, val_b, val_c)

print('Print results')
t2 = datetime.datetime.now()
while not q.empty():
    print(q.get())

t = t2 - t1
print('total time:', str(t.total_seconds()), 'seconds')
```

Save into a file, called ch11_08.py.

```
$ python3 ch11_08.py
```

Program output:

11.6.2 ProcessPoolExecutor

ProcessPoolExecutor uses process to do parallel tasks.

We implement the same scenario from previous section.

```
import multiprocessing
import concurrent.futures
import random
import time
import datetime


def perform(q, a, b, c):
    rand_val = random.uniform(0, 2)
    res = a * b * 10 - c * 2
    time.sleep(rand_val)

    q.put(res)


t1 = datetime.datetime.now()
m = multiprocessing.Manager()
q = m.Queue()
with concurrent.futures.ProcessPoolExecutor(max_workers=3) as executor:
    for i in range(1, 15):
        val_a = random.randint(1, 10)
```

```
        val_b = random.randint(1, 10)
        val_c = random.randint(1, 10)
        executor.submit(perform, q, val_a, val_b, val_c)

print('Print results')
t2 = datetime.datetime.now()
while not q.empty():
    print(q.get())

t = t2 - t1
print('total time:', str(t.total_seconds()), 'seconds')
```

Save these scripts into a file, called ch11_09.py.

```
$ python3 ch11_09.py
```

Program output:

12. Encoding

This chapter explains how to work with encoding in Python.

12.1 Getting Started

In this chapter, we explore encoding package from Python. The following is a list of our demo to illustrate how to use encoding package:

- Base64
- Hexadecimal
- JSON
- XML
- CSV

Let’s start to implement these encoding.

12.2 Encoding Base64

The first demo is to work with base64 encoding. We can use base64 package, https://docs.python.org/3/library/base64.html . To encode string to base64 string, we can use b64encode(). Otherwise, we can decode it using b64decode() function.

For testing, we encode a string message to base64. Then, we decode base64 message to original message. Write these scripts.

```
import base64


plaintext = 'Hello world from Python'
s_bytes = plaintext.encode()
enc1 = base64.b64encode(s_bytes)
dec1 = base64.b64decode(enc1)
s_dec1 = dec1.decode()

print('Plaintext:', plaintext)
print('Base64:', enc1)
print('Decoded:', s_dec1)
```

Save into a file, called ch12_01.py.

Now you can test to build and run the program.

```
$ python3 ch12_01.py
```

A sample output can be seen in Figure below.

12.3 Hexadecimal

The second demo is to encode and decode string to Hexadecimal. We can use encode and decode from Python 3.x, to implement our demo.

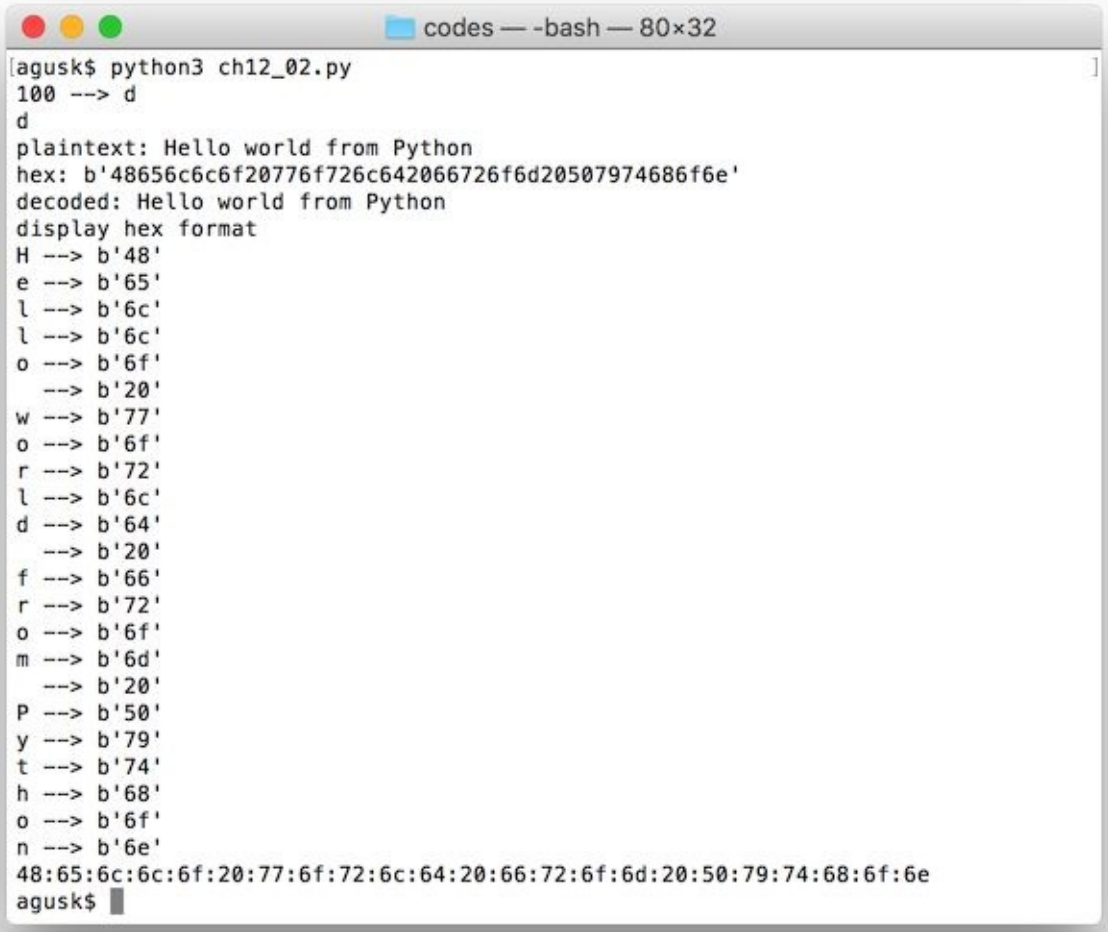
Create a file, called ch12_02.py. The following is implementation of encoding/decoding Hexadecimal.

```
from codecs import encode, decode

# declare hex data
num = 0x64
print(num, '-->', chr(num))
num_s = "\x64"
print(num_s)

# display hex from string data
s = 'Hello world from Python'
s_bytes = s.encode()
s_hex = encode(s_bytes,'hex')
s_decoded = decode(s_hex, 'hex')
s_plaintext = s_decoded.decode()

print('plaintext:', s)
print('hex:', s_hex)
print('decoded:', s_plaintext)

# samples for displaying hex data
print('display hex format')
for c in s:
    print(c,'-->', encode(c.encode(),'hex'))


hex2 = ":".join("{:02x}".format(c) for c in s_bytes)
print(hex2)
```

Save this code. Now you can build and run this program.

```
$ python3 ch12_02.py
```

A sample output can be seen in Figure below.


codes — -bash — 80×32
agusk$ python3 ch12_02.py
100 --> d
d
plaintext: Hello world from Python
hex: b'48656c6c6f20776f726c642066726f6d20507974686f6e'
decoded: Hello world from Python
display hex format
H --> b'48'
e --> b'65'
l --> b'6c'
l --> b'6c'
o --> b'6f'
--> b'20'
w --> b'77'
o --> b'6f'
r --> b'72'
l --> b'6c'
d --> b'64'
--> b'20'
f --> b'66'
r --> b'72'
o --> b'6f'
m --> b'6d'
--> b'20'
P --> b'50'
y --> b'79'
t --> b'74'
h --> b'68'
o --> b'6f'
n --> b'6e'
48:65:6c:6c:6f:20:77:6f:72:6c:64:20:66:72:6f:6d:20:50:79:74:68:6f:6e
agusk$

12.4 JSON

The third demo is to construct and parse JSON data. In Python, we can use json package, https://docs.python.org/3/library/json.html .

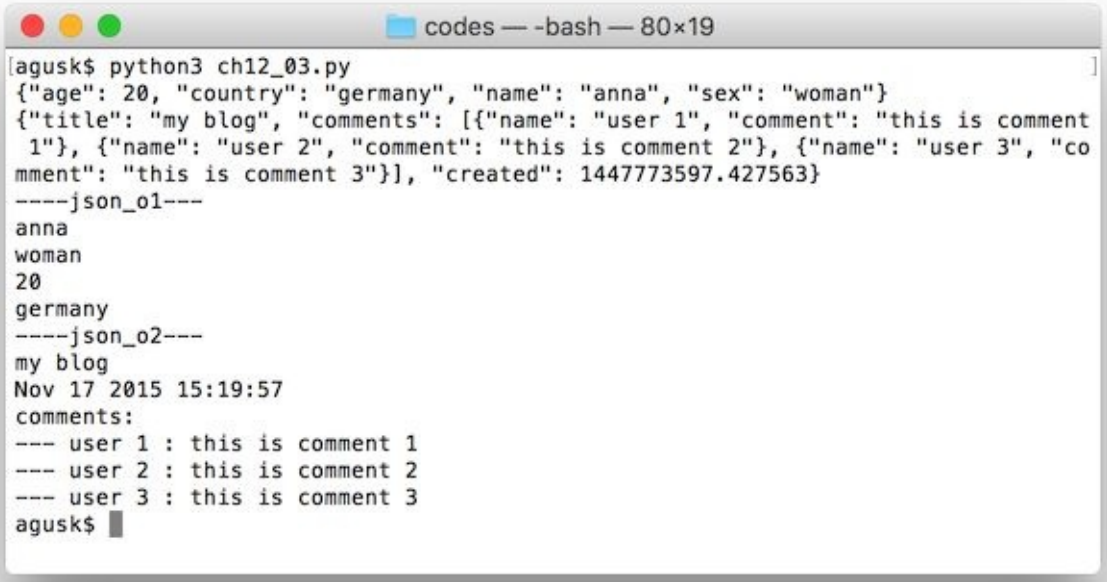
For demo, create a file, called ch12_02.py and write this code.

```
import json
import time


# construct json
data = {
    'name': 'anna',
    'sex': 'woman',
    'age': 20,
    'country': 'germany'
}
blog = {
    'title': 'my blog',
    'created': time.time(),
    'comments': [
        {'name':'user 1', 'comment': 'this is comment 1'},
        {'name':'user 2', 'comment': 'this is comment 2'},
        {'name':'user 3', 'comment': 'this is comment 3'}
    ]
}

# json object to json string
json_data = json.dumps(data)
json_data2 = json.dumps(blog)
print(json_data)
print(json_data2)

# decode json string to json object
# you define json string or load json string from file
json_o1 = json.loads(json_data)
json_o2 = json.loads(json_data2)

# iteration json values
print('----json_o1---')
print(json_o1['name'])
print(json_o1['sex'])
print(json_o1['age'])
print(json_o1['country'])

print('----json_o2---')
print(json_o2['title'])
created_s = time.strftime("%b %d %Y %H:%M:%S", time.gmtime(json_o2['crea
print(created_s)
print('comments:')
for comment in json_o2['comments']:
    print('---',comment['name'],':',comment['comment'])
```

Save this code. Now you can build and run this program.

```
$ python3 ch12_03.py
```

A sample output can be seen in Figure below.



12.5 XML

The fourth demo is to read and write XML data. We can use xml package, https://docs.python.org/3/library/xml.html . In this demo, we use xml.etree.ElementTree to process XML data. In this demo, we read xml file and create a new xml file.

Firstly, we create xml file for testing, called products.xml and write this data.

```
<?xml version="1.0"?>
<products>
    <product name="product 1">
        <code>1001</code>
        <year>2015</year>
        <model color="red" category="food">12E</model>
    </product>
    <product name="product 2">
        <code>1002</code>
        <year>2015</year>
        <model color="green" category="beverage">15C</model>
    </product>
    <product name="product 3">
        <code>1003</code>
        <year>2015</year>
        <model color="blue" category="electronics">19A</model>
    </product>
</products>
```

Now we read xml file and display it into Terminal.

```
import xml.etree.ElementTree as Et

# load xml file and iterate
xml_tree = Et.parse('products.xml')
products = xml_tree.getroot()
print(products.tag)
for product in products:
    print(' ',product.tag, ' name=',product.get('name'))
    for product_item in product:
        print('    ',product_item.tag, '=',product_item.text)

# finding specific data
print('----------')
for code in products.iter('code'):
    print(code.text)

# construct xml and save into a file
print('construct xml file')
users = Et.Element('users')
for i in range(1, 5):
    user = Et.SubElement(users, 'user')
```

```
    user.set('name', "User " + str(i))

    user_item = Et.SubElement(user, 'age')
    user_item.text = str(i * 3)

    user_item2 = Et.SubElement(user, 'id')
    user_item2.text = "1203" + str(i)


print('write into xml file')
tree = Et.ElementTree(users)
tree.write("users.xml")
```

Save this code into a fiile, called ch12_04.py. Now you can run this program.

```
$ python3 ch12_04.py
```

If success, you see users.xml file and you should see the content as follows.

```
<users>
<user name="User 1"><age>3</age><id>12031</id></user>
<user name="User 2"><age>6</age><id>12032</id></user>
<user name="User 3"><age>9</age><id>12033</id></user>
<user name="User 4"><age>12</age><id>12034</id></user>
</users>
```

A sample of program output can be seen in Figure below.

12.6 CSV

The last demo is to read and write data CSV which is a collection of comma-separated data. We can access CSV file using csv package, https://docs.python.org/3/library/csv.html . Now we build a program to read csv file and write data into csv file.

For testing, we create a CSV file, **customers.csv**, with the following content.

```
id,full_name,age,country
12,James Butt,23,US
13,Josephine Darakjy,40,UK
14,Art Venere,35,US
15,Lenna Paprocki,34,DE
16,Donette Foller,27,NL
```

The following is implementation of reading/writing CSV file.

```
import csv

# reading csv file
with open('customers.csv', newline='') as csv_file:
    customers = csv.reader(csv_file, delimiter=',')
    for row in customers:
        print(', '.join(row))
csv_file.close()

print('------------')
# reading csv file with handling header
with open('customers.csv') as csv_file:
    reader = csv.DictReader(csv_file)
    for row in reader:
        print(row['id'], row['full_name'], row['age'], row['country'])
csv_file.close()


# writing csv file
print('----------------')
print('writing csv file')
with open('cities.csv', 'w') as csv_file:
    fieldnames = ['id', 'name', 'country']
    writer = csv.DictWriter(csv_file, fieldnames=fieldnames, delimiter='

    writer.writeheader()
    for i in range(1,10):
        writer.writerow({'id': i, 'name': "city " + str(i), 'country': "

csv_file.close()
print('done')
```

Note: You can change CSV file path.

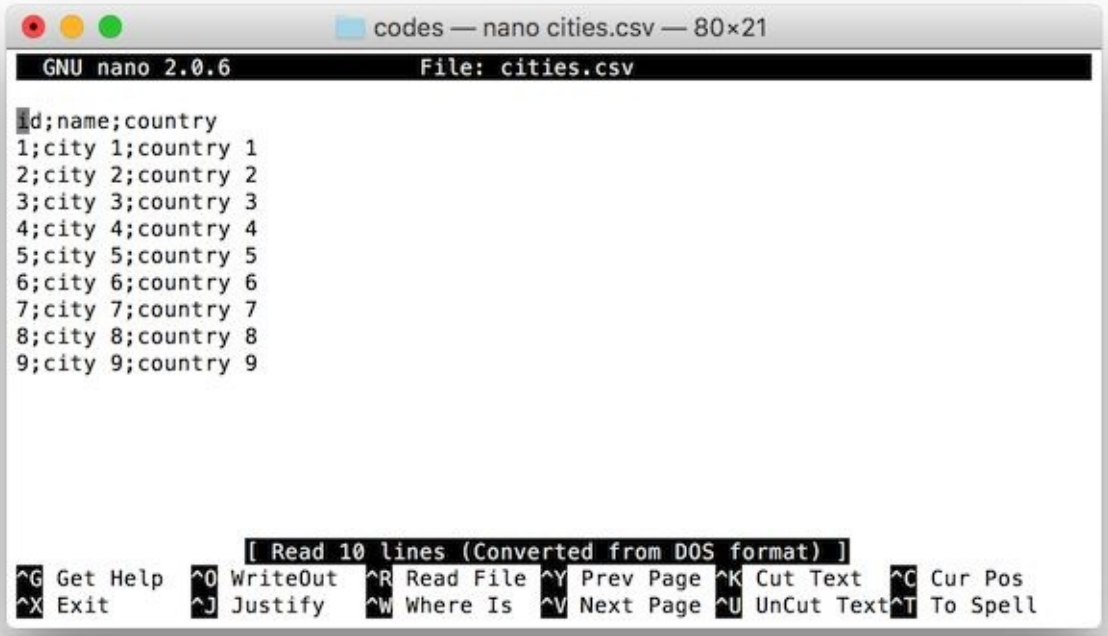
Save this code into a file, called ch12_05.py. Now you can run this program.

```
$ python3 ch12_05.py
```

A sample output can be seen in Figure below.

The program also generate cities.csv file.

If you open **cities.csv**, you get a content of cities data like a content of **customers.csv** file.


codes — nano cities.csv — 80×21
GNU nano 2.0.6 File: cities.csv
id;name;country
1;city 1;country 1
2;city 2;country 2
3;city 3;country 3
4;city 4;country 4
5;city 5;country 5
6;city 6;country 6
7;city 7;country 7
8;city 8;country 8
9;city 9;country 9
[Read 10 lines (Converted from DOS format)]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell

13. Hashing and Cryptography

This chapter explains how to work with hashing and cryptography in Python.

13.1 Getting Started

Hashing is generating a value or values from a string of text using a mathematical function. Cryptography is the practice and study of techniques for secure communication in the presence of third parties (called adversaries), http://en.wikipedia.org/wiki/Cryptography . In this chapter, I don’t explain mathematical hashing and Cryptography. You can read those materials on textbooks.

In this chapter, we explore how to work with hashing implementation using Python. The following is hashing algorithms which we use in this book:

- MD5
- SHA1 and SHA256
- Hashing with Key (HMAC)

The next topic is to implement Cryptography using Python. We explore symmetric and asymmetric Cryptography.

13.2 Hashing

Basically, you can explore how to implement hashing or hash function using Python via https://docs.python.org/3/library/hashlib.html. We also use open source library, called pycrypto, https://pypi.python.org/pypi/pycrypto . We implement both in our case.

In this section, we explore several hashing algorithms, for instance, MD5, SHA1, SHA256 and HMAC.

13.2.1 Hashing with MD5

We can use MD5 using md5 package, https://docs.python.org/3/library/hashlib.html. To calculate a hash value from a text , we can call digest() function.

For illustration, we do hashing a plaintext.

```
import hashlib
import binascii

plaintext = 'hello world from python'

# md5
md5 = hashlib.md5()
md5.update(plaintext.encode())
hash_md5 = md5.digest()
hex_hash_md5 = md5.hexdigest()
print('hash md5:', hash_md5)
print('hex hash md5:', hex_hash_md5)
```

digest() is used to calculate a hash value. hexdigest() is to calculate hex hash.

13.2.2 Hashing with SHA1 and SHA256

The second demo is to implement hash function using sha1 and sha256.

For illustration, write these script for hashing SHA1 and SHA256.

```
import hashlib
import binascii

plaintext = 'hello world from python'


# sha1
sha1 = hashlib.sha1()
```

```
sha1.update(plaintext.encode())
hash_sha1 = sha1.digest()
hex_hash_sha1 = sha1.hexdigest()
print('hash sha1:', hash_sha1)
print('hex hash sha1:', hex_hash_sha1)

# sha256
sha256 = hashlib.sha256()
sha256.update(plaintext.encode())
hash_sha256 = sha256.digest()
hex_hash_sha256 = sha256.hexdigest()
print('hash sha256:', hash_sha256)
print('hex hash sha256:', hex_hash_sha256)
```

13.2.3 Hashing with Key Using HMAC

A keyed-hash message authentication code (HMAC) is a specific construction for calculating a message authentication code (MAC) involving a cryptographic hash function in combination with a secret cryptographic key, http://en.wikipedia.org/wiki/Hash-based_message_authentication_code . In Python, we use pbkdf2_hmac() object.

For illustration, we do hashing a plaintext with key.

```
import hashlib
import binascii

plaintext = 'hello world from python'

# hash with key
# hmac
key = 'p4ssw0rd'
hmac = hashlib.pbkdf2_hmac('sha256', key.encode(), plaintext.encode(), 1
hex_hash_hmac = binascii.hexlify(hmac)
print('hex hash hmac:', hex_hash_hmac)
```

13.2.4 Write them All

Save all code for our demo on this section. Write this code and save into a file, called ch13_01.py.

```
import hashlib
import binascii

plaintext = 'hello world from python'

# md5
```

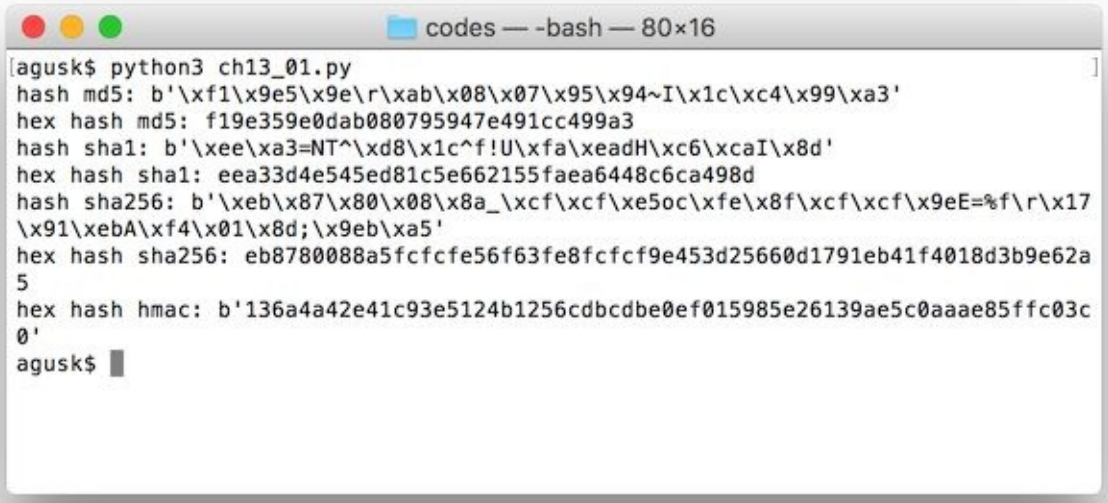
```
md5 = hashlib.md5()
md5.update(plaintext.encode())
hash_md5 = md5.digest()
hex_hash_md5 = md5.hexdigest()
print('hash md5:', hash_md5)
print('hex hash md5:', hex_hash_md5)

# sha1
sha1 = hashlib.sha1()
sha1.update(plaintext.encode())
hash_sha1 = sha1.digest()
hex_hash_sha1 = sha1.hexdigest()
print('hash sha1:', hash_sha1)
print('hex hash sha1:', hex_hash_sha1)

# sha256
sha256 = hashlib.sha256()
sha256.update(plaintext.encode())
hash_sha256 = sha256.digest()
hex_hash_sha256 = sha256.hexdigest()
print('hash sha256:', hash_sha256)
print('hex hash sha256:', hex_hash_sha256)

# hash with key
# hmac
key = 'p4ssw0rd'
hmac = hashlib.pbkdf2_hmac('sha256', key.encode(), plaintext.encode(), 1
hex_hash_hmac = binascii.hexlify(hmac)
print('hex hash hmac:', hex_hash_hmac)
```

Save all.

Now you can test to build and run the program.

```
$ python3 ch13_01.py
```

A sample output can be seen in Figure below.

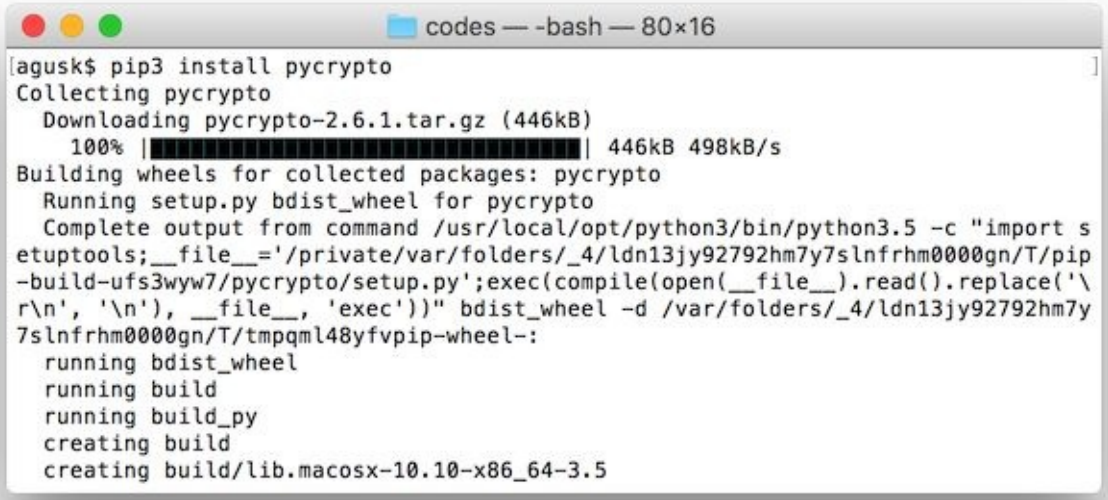

codes — -bash — 80×16
agusk$ python3 ch13_01.py
hash md5: b'\xf1\x9e5\x9e\r\xab\x08\x07\x95\x94~I\x1c\xc4\x99\xa3'
hex hash md5: f19e359e0dab080795947e491cc499a3
hash sha1: b'\xee\xa3=NT^\xd8\x1c^f!U\xfa\xeadH\xc6\xcaI\x8d'
hex hash sha1: eea33d4e545ed81c5e662155faea6448c6ca498d
hash sha256: b'\xeb\x87\x80\x08\x8a_\xcf\xcf\xe5oc\xfe\x8f\xcf\xcf\x9eE=%f\r\x17
\x91\xebA\xf4\x01\x8d;\x9eb\xa5'
hex hash sha256: eb8780088a5fcfcfe56f63fe8fcfcf9e453d25660d1791eb41f4018d3b9e62a
5
hex hash hmac: b'136a4a42e41c93e5124b1256cdbcdbe0ef015985e26139ae5c0aaae85ffc03c
0'
agusk$

13.3 Cryptography

In this section, we focus Symmetric and Asymmetric Cryptography. In Symmetric Cryptography, we use the same key to encrypt and decrypt. Otherwise, Asymmetric Cryptography uses different key to encrypt and decrypt.

We use pycrypto, https://github.com/dlitz/pycrypto . You can install it via pip. A sample of command to install pycrypto for Python 3.x.

```
$ pip3 install pycrypto
```



For illustration, we hash a plaintext using SHA256. Write these scripts and save into a file, called ch13_02.py.

```
from Crypto.Hash import SHA256

plaintext = 'hello world from python'

sha256 = SHA256.new()
sha256.update(plaintext.encode())
hash_sha256 = sha256.digest()
hex_hash_sha256 = sha256.hexdigest()
print('hash sha256:', hash_sha256)
print('hex hash sha256:', hex_hash_sha256)
```

Save and run the program.

```
$ python3 ch13_02.py
```

Program output:

13.3.1 Symmetric Cryptography

There are many algorithms to implement Symmetric Cryptography. In this section, we use AES algorithm. The Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001,
http://en.wikipedia.org/wiki/Advanced_Encryption_Standard.

We can do symmetric Cryptography using pycrypto by writing these scripts into a file, called ch13_03.py

```
from Crypto import Random
from Crypto.Cipher import AES

message = 'hello world from python'
# AES key must be either 16, 24, or 32 bytes long
key = 'p4ssw0rdp4ssw0rd'
print('message:', message)

# encrypt
iv_aes = Random.new().read(AES.block_size)
cipher_aes = AES.new(key.encode(), AES.MODE_CFB, iv_aes)
encrypted_aes = cipher_aes.encrypt(message.encode())
print('encrypted AES:', encrypted_aes)


# decrypted
dec_iv_aes = Random.new().read(AES.block_size)
dec_cipher_aes = AES.new(key.encode(), AES.MODE_CFB, iv_aes)
```

```
decrypted_aes = dec_cipher_aes.decrypt(encrypted_aes)
print('decrypted AES:', decrypted_aes)
```

Explanation:

The following is the steps for encryption

- Define a key. It should be 16, 24, or 32 key length
- Calculate IV value for AES using Random.read() with AES.block_size parameter
- Instantiate AES using AES.new() with passing key and IV value
- Encrypt message by calling encrypt()
- The result is array of byte

The following is the steps for encryption

- Define a key. It should be 16, 24, or 32 key length
- Calculate IV value for AES using Random.read() with AES.block_size parameter
- Instantiate AES using AES.new() with passing key and IV value
- Decrypt cipher by calling decrypt()
- The result is be string in array of byte

Save and run the program.

```
$ python3 ch13_03.py
```

Program output:

13.3.2 Asymmetric Cryptography

The common algorithm to implement Asymmetric Cryptography is RSA which is widely used for secure data transmission. You read a brief description in Wikipedia, http://en.wikipedia.org/wiki/RSA_(cryptosystem) .

pycrypto library has library for RSA implementation. In this section, we try to implement RSA using pycrypto . The following is our scenario:

- Generate RSA keys (public and private keys)
- Save these keys to two files (public and private key files)
- For encryption, we use public key file
- For decryption, we use private key file

We store public and private keys into a file in PEM data encoding.

To generate public and private keys for RSA, we use RSA.generate(). We extract private and public key values. Then, save them into file. encrypt() and decrypt() from RSAKey are used to encrypt and decrypt.

The following is implementation for our RSA scenario.

```
from Crypto.PublicKey import RSA
from Crypto import Random

# generate private and public keys
# Then, save them into files
print('generating private and public keys…')
key = RSA.generate(2048)
f = open('my_rsa_private_key.pem', 'wb')
f.write(key.exportKey('PEM'))
f.close()
f = open('my_rsa_public_key.pem', 'wb')
f.write(key.publickey().exportKey('PEM'))
f.close()
print('done')


message = 'hello world from python'
print('plaintext:', message)

# encrypt data using public key
f = open('my_rsa_public_key.pem','r')
RSAkey = RSA.importKey(f.read())
f.close()
k = Random.new().read(8)
encrypted_msg = RSAkey.encrypt(message.encode(), k)
ciphertext = encrypted_msg[0]
print('encrypted:', ciphertext)
```

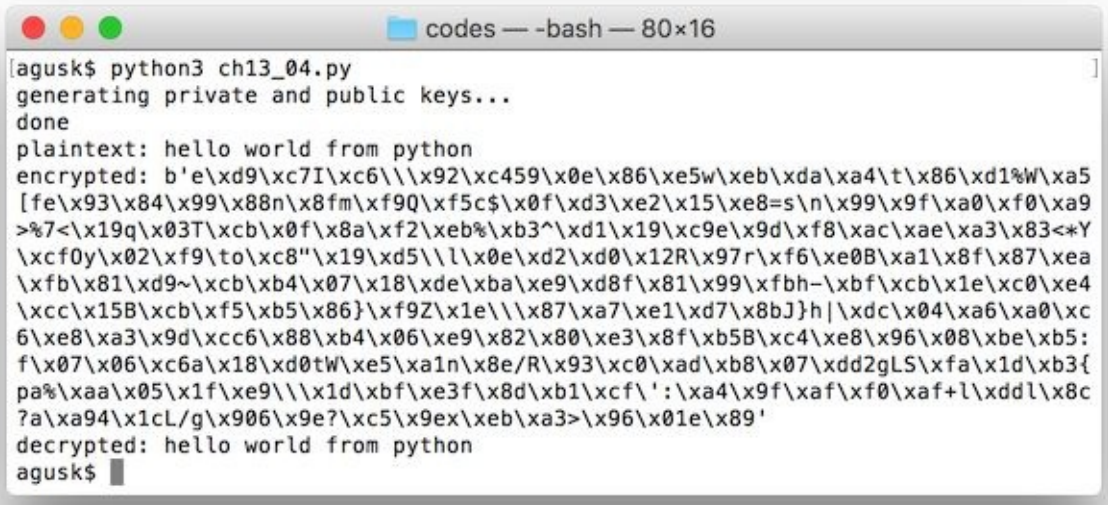
```
# decrypt data using private key
f = open('my_rsa_private_key.pem','r')
RSAkey = RSA.importKey(f.read())
f.close()
decrypted_msg = RSAkey.decrypt(ciphertext)
print('decrypted:', decrypted_msg.decode("utf-8"))
```

Save these scripts into a file, called ch13_04.py.

Now you can build and run it. A sample output can be seen in Figure below.

```
$ python3 ch13_04.py
```

Program output:



14. Database Programming

This chapter explains how to build database application using Python.

14.1 Database for Python

Python can communicate with database server through database driver. We can use MySQL driver fro Python. In this chapter, I only focus on MySQL scenario.

14.2 MySQL Driver for Python

We use MySQL driver for Python. Further information about this driver, please visit on https://dev.mysql.com/downloads/connector/python/2.1.html to download and install.

You also can install MySQL driver for Python via pip3 (Python 3.x).

```
$ sudo pip3 install --allow-external mysql-connector-python mysql-connec
```

14.3 Testing Connection

In this section, we try to connect MySQL database. We can use connect() from mysql.connector object.

```
import mysql.connector

print('connecting to mysql server…')
cnx = mysql.connector.connect(user='pyuser',
                              password='password123',
                              host='127.0.0.1',
                              database='pydb')
print('connected')
```

You should pass username, password, database server and database name.

14.4 CRUD (Create, Read, Update and Delete) Operations

In this section, we try to create, read, update and delete data on MySQL. Firstly, we create database and its table.

The following is our table scheme on MySQL.

```
CREATE DATABASE `pydb`;

CREATE TABLE `pydb`.`product` (
  `idproduct` INT NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(30) NOT NULL,
  `code` VARCHAR(10) NOT NULL,
  `price` DECIMAL NOT NULL,
  `quantity` INT NULL,
  `created` DATETIME NULL,
  PRIMARY KEY (`idproduct`));
```

File: pydb.sql

Run these SQL scripts into your MySQL.

14.4.1 Create Data

To create data, we use SQL statement, INSERT INTO, which pass to execute() function.

```
def create_data(conn):
    cursor = conn.cursor()
    print('inserting data…')
    for i in range(1,5):
        insert_product = ("INSERT INTO product "
                  "(name, code, price, quantity, created) "
                  "VALUES (%s, %s, %s, %s, %s)")
        data_product = ("product " + str(i), "F029" + str(i), i*0.21, i,
        cursor.execute(insert_product, data_product)
        product_id = cursor.lastrowid
        print('inserted with id=',product_id)

    conn.commit()
    cursor.close()
    print('done')
```

To obtain the last inserted id, we can use lastrowid from cursor object.

14.4.2 Read Data

To read data, you can use SELECT…FROM query on your Python scripts.

```
def read_data(conn):
    print('reading data….')
    selected_id = 0
    cursor = conn.cursor()
    query = "SELECT idproduct, name, code, price, quantity, created FROM
    cursor.execute(query)
    for (id, name, code, price, quantity, created) in cursor:
        print("{}, {}, {}, {}, {}, {:%d %b %Y %H:%M:%S}".format(
                id, name, code, price, quantity, created))
        if selected_id <= 0:
            selected_id = id

    cursor.close()
    print('done')

    return selected_id
```

14.4.3 Update Data

To update data, you can use UPDATE….SET query on your Python scripts.

```
def update_data(conn, id):
    print('updating data with idproduct=', id, '...')
    cursor = conn.cursor()
    query = "UPDATE product SET name=%s, code=%s, price=%s, quantity=%s,
    name = 'updated-name'
    code = 'F9999'
    price = 0.99
    quantity = 10
    created = datetime.now()
    cursor.execute(query, (name, code, price, quantity, created, id))
    conn.commit()
    cursor.close()
    print('done')
```

Don’t forget to call commit() after changed the data.

14.4.4 Delete Data

To update data, you can use DELETE FROM query on your Python scripts.

```
def delete_data(conn, id):
    print('deleting data on idproduct=', id, '...')
    cursor = conn.cursor()
    query = "DELETE FROM product where idproduct = %s "
    cursor.execute(query, (id,))
    conn.commit()
    cursor.close()
    print('done')


def delete_all(conn):
    print('deleting all data….')
    cursor = conn.cursor()
    query = "DELETE FROM product"
    cursor.execute(query)
    conn.commit()
    cursor.close()
    print('done')
```

Don’t forget to call commit() after changed the data.

14.4.5 Write them All

Now we can write our scripts about CRUD.

Write these scripts.

```
import mysql.connector
from datetime import datetime


def create_data(conn):
    cursor = conn.cursor()
    print('inserting data…')
    for i in range(1,5):
        insert_product = ("INSERT INTO product "
                  "(name, code, price, quantity, created) "
                  "VALUES (%s, %s, %s, %s, %s)")
        data_product = ("product " + str(i), "F029" + str(i), i*0.21, i,
        cursor.execute(insert_product, data_product)
        product_id = cursor.lastrowid
        print('inserted with id=',product_id)

    conn.commit()
    cursor.close()
    print('done')


def read_data(conn):
```

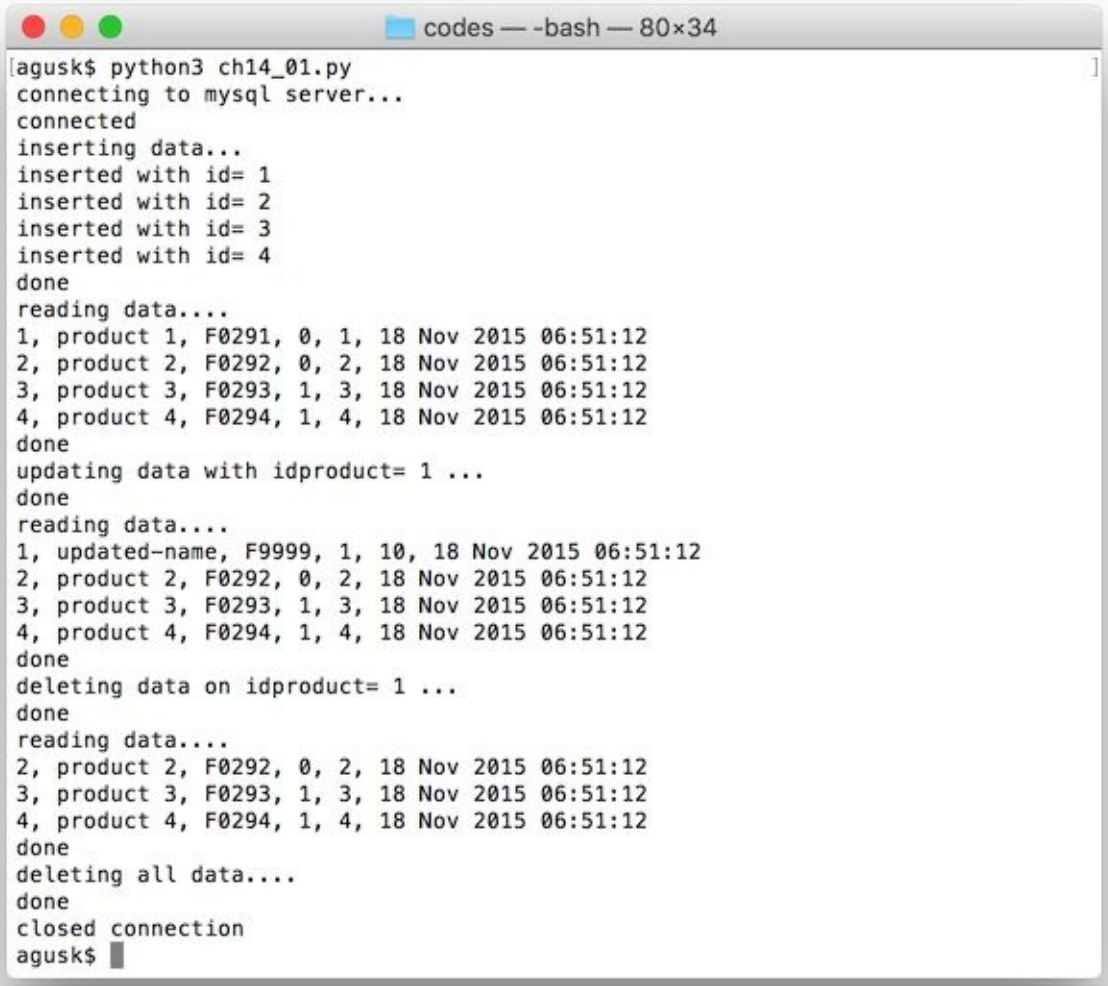
```
    print('reading data….')
    selected_id = 0
    cursor = conn.cursor()
    query = "SELECT idproduct, name, code, price, quantity, created FROM
    cursor.execute(query)
    for (id, name, code, price, quantity, created) in cursor:
        print("{}, {}, {}, {}, {}, {:%d %b %Y %H:%M:%S}".format(
                id, name, code, price, quantity, created))
        if selected_id <= 0:
            selected_id = id

    cursor.close()
    print('done')

    return selected_id


def update_data(conn, id):
    print('updating data with idproduct=', id, '...')
    cursor = conn.cursor()
    query = "UPDATE product SET name=%s, code=%s, price=%s, quantity=%s,
    name = 'updated-name'
    code = 'F9999'
    price = 0.99
    quantity = 10
    created = datetime.now()
    cursor.execute(query, (name, code, price, quantity, created, id))
    conn.commit()
    cursor.close()
    print('done')


def delete_data(conn, id):
    print('deleting data on idproduct=', id, '...')
    cursor = conn.cursor()
    query = "DELETE FROM product where idproduct = %s "
    cursor.execute(query, (id,))
    conn.commit()
    cursor.close()
    print('done')


def delete_all(conn):
    print('deleting all data….')
    cursor = conn.cursor()
    query = "DELETE FROM product"
    cursor.execute(query)
    conn.commit()
    cursor.close()
    print('done')


print('connecting to mysql server…')
```

```
cnx = mysql.connector.connect(user='pyuser',
                              password='password123',
                              host='127.0.0.1',
                              database='pydb')
print('connected')

create_data(cnx)
selected_id = read_data(cnx)
update_data(cnx, selected_id)
read_data(cnx)
delete_data(cnx, selected_id)
read_data(cnx)
delete_all(cnx)

cnx.close()
print('closed connection')
```

Save this program. Then, you can run it.

```
$ python3 ch14_01.py
```

Program output:


codes — -bash — 80×34
agusk$ python3 ch14_01.py
connecting to mysql server...
connected
inserting data...
inserted with id= 1
inserted with id= 2
inserted with id= 3
inserted with id= 4
done
reading data....
1, product 1, F0291, 0, 1, 18 Nov 2015 06:51:12
2, product 2, F0292, 0, 2, 18 Nov 2015 06:51:12
3, product 3, F0293, 1, 3, 18 Nov 2015 06:51:12
4, product 4, F0294, 1, 4, 18 Nov 2015 06:51:12
done
updating data with idproduct= 1 ...
done
reading data....
1, updated-name, F9999, 1, 10, 18 Nov 2015 06:51:12
2, product 2, F0292, 0, 2, 18 Nov 2015 06:51:12
3, product 3, F0293, 1, 3, 18 Nov 2015 06:51:12
4, product 4, F0294, 1, 4, 18 Nov 2015 06:51:12
done
deleting data on idproduct= 1 ...
done
reading data....
2, product 2, F0292, 0, 2, 18 Nov 2015 06:51:12
3, product 3, F0293, 1, 3, 18 Nov 2015 06:51:12
4, product 4, F0294, 1, 4, 18 Nov 2015 06:51:12
done
deleting all data....
done
closed connection
agusk$

15. Socket Programming

This chapter explains how to create socket application using Python.

15.1 Socket Module

We can create application based on socket stack using net package. You can find it for further information on https://docs.python.org/3/library/socket.html . I recommend you to read some books or websites about the concept of socket.

15.2 Hello World

To get started, we create a simple application to get a list of IP Address in local computer. We can use gethostname() and gethostbyname() from socket object.

In this section, we try to get local IP Address. Firstly, we can create a new file, called ch15_01.py.

```
import socket


hostname = socket.gethostname()
ip = socket.gethostbyname(hostname)
print('hostname:', hostname)
print('ip address:', ip)
```

Save this code. Try to build and run it.

```
$ python3 ch15_01.py
```

You should see your local IP address from your computer.

15.3 Client/Server Socket

Now we create a client/server socket using Python. We will use socket package to build client/server application. For illustration, we create server and client.

15.3.1 Server Socket

How to create server socket? It is easy. The following is a simple algorithm how to build server socket

- create server socket
- listen incoming client on the specific port
- if client connected, server sends data and then disconnect from client

In this section, we build server app. Firstly, we can create a new file, called ch15_02.py. Then, write these scripts.

```
import socket


# create tcp/ip socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

host = socket.gethostname()
port = 8091

# bind to the port
server.bind((host, port))
# queue up to 10 clients
server.listen(10)
counter = 0
print('waiting connection from clients…')
while True:
    # establish a connection
    client, address = server.accept()

    counter += 1
    print('a new connection from',str(address))
    message = "welcome, your id=" + str(counter) + "\r\n"
    client.send(message.encode('ascii'))
    client.close()
```

It uses port 8091. You can change it.

15.3.2 Client Socket

Client socket is client application that connects to server and then sends/receives data from/to server. We should know about IP address and port from target server. We can call connect() to connect to server and call recv() to receive incoming data.

In this section, we build client app. Firstly, we can create a new file, called ch15_03.py. Then, write these scripts.

```
import socket

# create a socket object
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# ip/hostname of server
# change this ip address
host = '192.168.0.10'
port = 8091

print('connecting to server…')
client.connect((host, port))
print('connected')

recv = client.recv(1024)
print('received:', recv.decode('ascii'))

client.close()
print('closed')
```

You should change host for IP address of server.

15.3.3 Testing

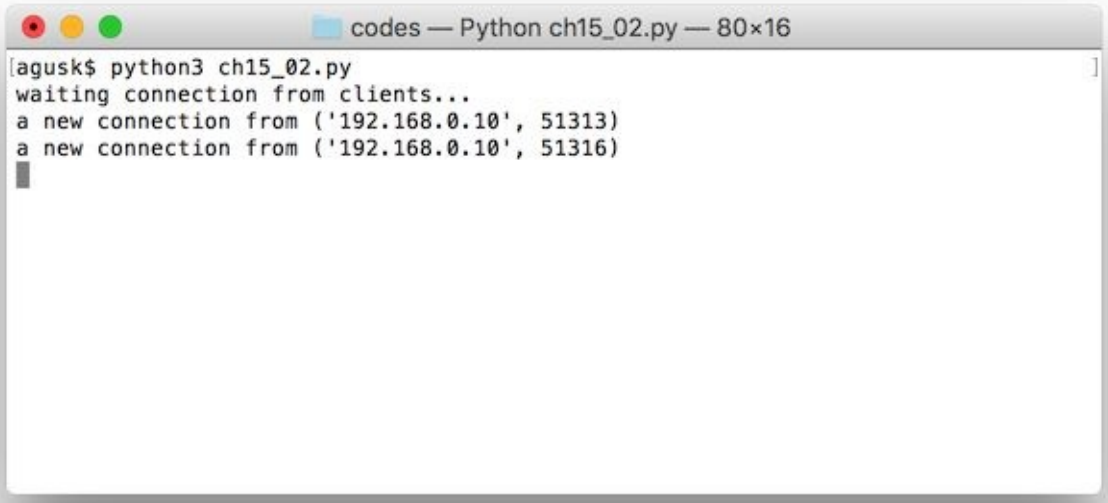
Now we can test our client/server application. Firstly, we run server application and then execute client application.

```
$ python3 ch15_02.py
```

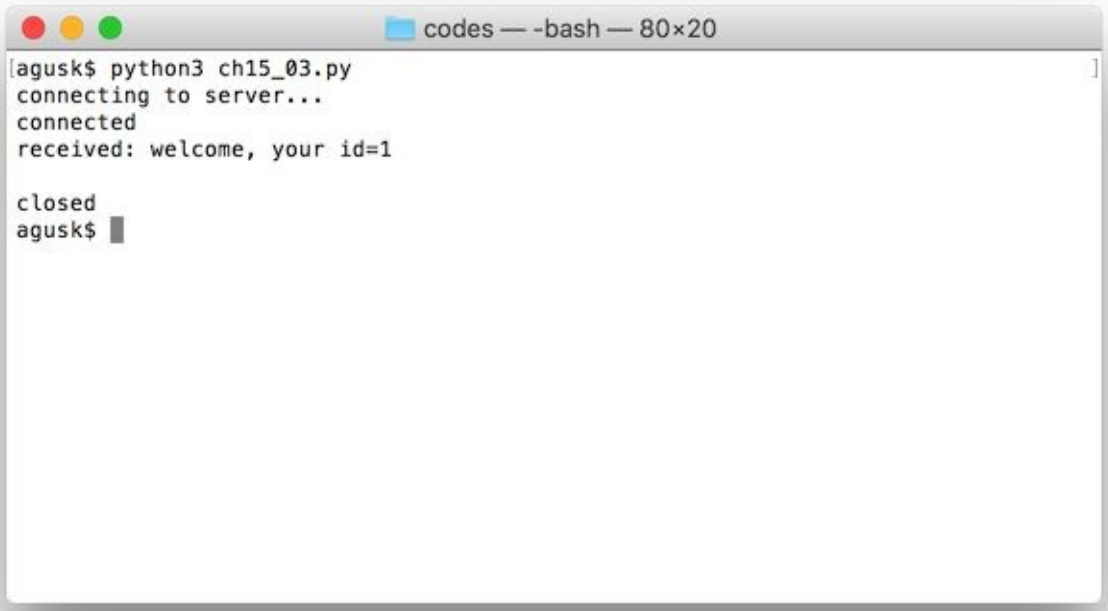
Then, run client app.

```
$ python3 ch15_03.py
```

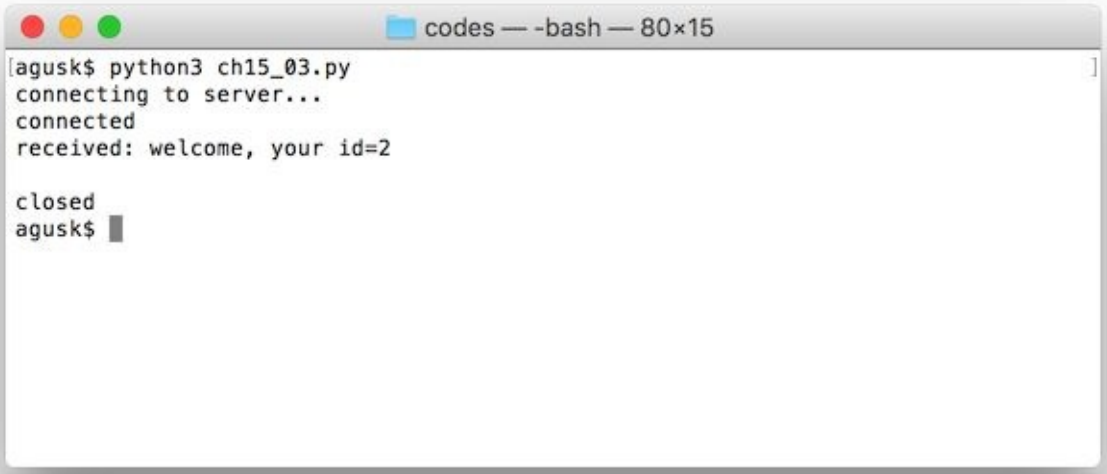
Here is sample of program output for server application:



Here is sample of program output for client application:



This is program output for the second client.


codes — -bash — 80×15
agusk$ python3 ch15_03.py
connecting to server...
connected
received: welcome, your id=2
closed
agusk$

16. Python Regular Expressions

This chapter explains how to work with regular expressions in Python.

16.1 Getting Started

Regular expressions are a powerful language for matching text patterns. The Python “re” module provides regular expression support. This library can be read on https://docs.python.org/3/library/re.html .

You also obtain regular expression patterns from this site, http://www.regxlib.com .

16.2 Demo

In this demo, we create three scenario:

- Validate number data
- Search data
- Search and replace data

We can use match() from re object to validate the matching of defined pattern. To search, you can use search() function and use pub() to replace data.

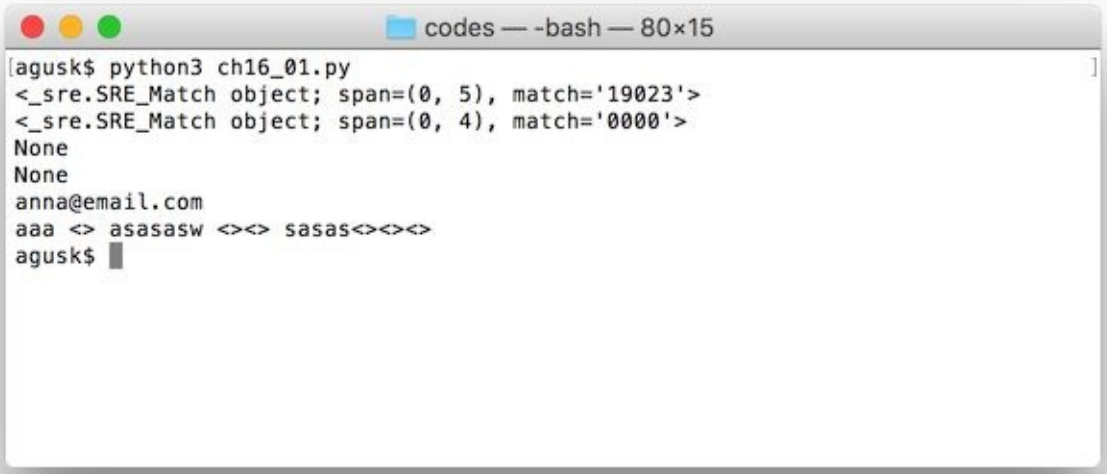
Let's create a file, called ch16_01.py, and write these scripts.

```
import re

# pattern for numbers
p = re.compile('^[0-9]+$')
print(p.match('19023'))
print(p.match('0000'))
print(p.match('12.789'))
print(p.match('12b23'))

# search
message = 'Anna William <anna@email.com>'
match = re.search(r'[\w.-]+@[\w.-]+', message)
if match:
    print(match.group())


# search and replase
message = 'aaa : asasasw :: sasas:::'
p = re.compile('(:|::|:::)')
resutl = p.sub('<>', message)
print(resutl)
```

Save and run the program.

```
$ python3 ch16_01.py
```

Program output:


codes — -bash — 80×15
agusk$ python3 ch16_01.py
<_sre.SRE_Match object; span=(0, 5), match='19023'>
<_sre.SRE_Match object; span=(0, 4), match='0000'>
None
None
anna@email.com
aaa <> asasasw <><> sasas<><><>
agusk$

17. Python GUI Programming

This chapter explains how to work with GUI in Python.

17.1 Getting Started

There are many modules to implement GUI in Python. In this chapter, we learn tkinter to build Python GUI. This library can be read on this site, https://docs.python.org/3/library/tk.html . This library has installed on Python 3.x.

Let’s start to build Python app with tkinter library.

17.2 Hello Python GUI

The first demo is to build Python GUI Hello World. We use Tk object to build a form.

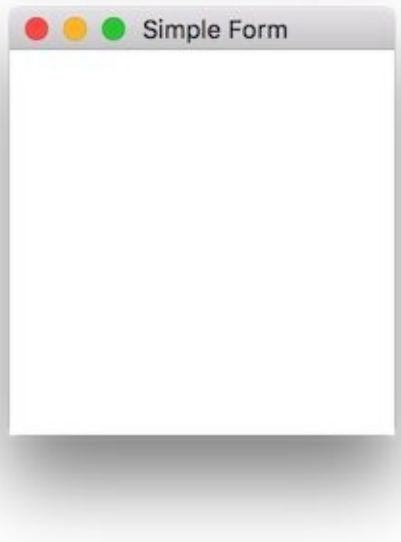
Write these scripts.

```
import tkinter as tk


dialog = tk.Tk()
dialog.title('Simple Form')
dialog.mainloop()
```

Save these scripts into a file, called ch17_01.py. Then, run the program.

```
$ python3 ch17_01.py
```

You should see a form dialog with title “Simple Form”.



17.3 Working with Input Form

Now we can extend our Tk object into an input form. In this scenario, we put two Textbox and a button. If we click a button, we read Textbox values.

Let’s write these scripts for demo.

```
import tkinter as tk


class InputForm(object):
    def __init__(self):
        self.root = tk.Tk()
        self.root.title('Input Form')
        self.num_a = ''
        self.num_b = ''
        self.frame = tk.Frame(self.root)
        self.frame2 = tk.Frame(self.root)
        self.frame.pack()
        self.frame2.pack()
        self.initialization()

    def initialization(self):
        r = self.frame
        k_a = tk.Label(r,text='Number A')
        k_a.grid(row=0, column=0)
        self.e_a = tk.Entry(r, text='NumA')
        self.e_a.grid(row=0, column=1)
        self.e_a.focus_set()
        k_b = tk.Label(r,text='Number B')
        k_b.grid(row=1, column=0)
        self.e_b = tk.Entry(r, text='NumB')
        self.e_b.grid(row=1, column=1)
        self.e_b.focus_set()
        r2 = self.frame2
        b = tk.Button(r2,text='Save',command=self.get_inputs)
        b.pack(side='left')

    def get_inputs(self):
        self.num_a = self.e_a.get()
        self.num_b = self.e_b.get()
        self.root.destroy()

    def get_values(self):
        return self.num_a, self.num_b

    def wait_for_input(self):
        self.root.mainloop()


dialog = InputForm()
dialog.wait_for_input()
num_a, num_b = dialog.get_values()
```

```
print('num a:', num_a)
print('num b:', num_b)
```

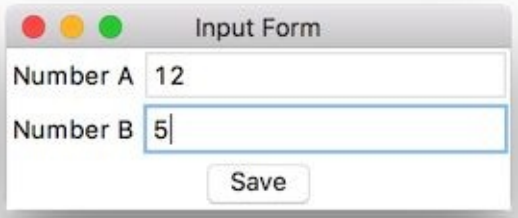
Save the program into a file, called ch17_02.py.

Run the program.

```
$ python3 ch17_02.py
```

You should see the input form dialog.

Fill values on Textbox. Then, click Save button.



After clicked, the program will read input values and shows them on Terminal.

codes — -bash — 80×15
agusk$ python3 ch17_02.py
num a: 12
num b: 5
agusk$

17.4 Working with Common Dialogs

tkinter library also provides common dialogs such Messagebox, filedialog and colorchooser.

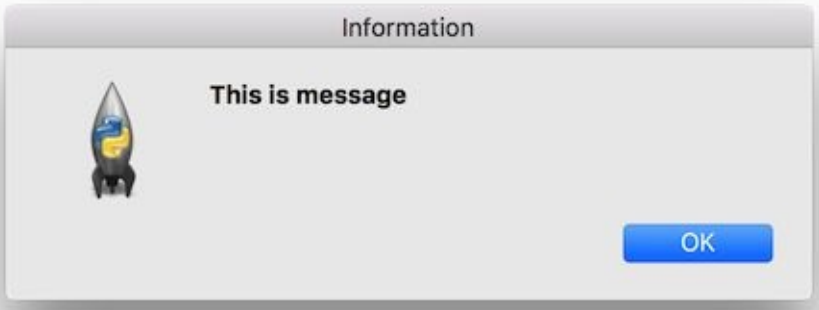
For illustration, write these scripts.

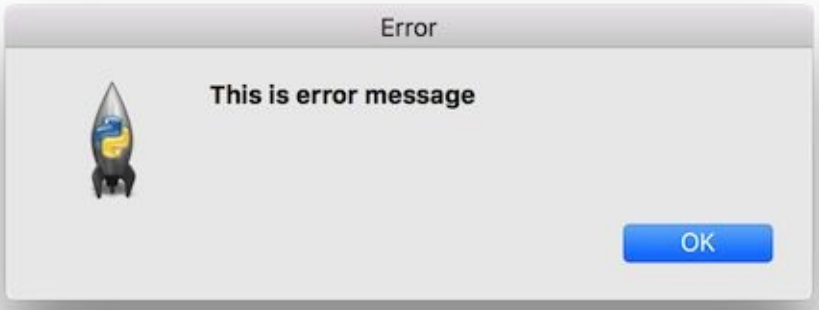
```
import tkinter as tk
from tkinter import messagebox, filedialog
from tkinter.colorchooser import *


# messagebox
print('demo messagebox')
messagebox.showinfo('Information', 'This is message')
messagebox.showerror('Error', 'This is error message')
messagebox.showwarning('Warning', 'This is warning message')

# filedialog
dir = filedialog.askdirectory()
print('selected directory:', dir)
file = filedialog.askopenfile(mode="r")
print('selected file:', file.name)
new_file_name = filedialog.asksaveasfilename()
print('save as file:', new_file_name)


# colorchooser
def get_color():
    color = askcolor()
    print('selected color:', color)

dialog = tk.Tk()
tk.Button(dialog, text='Select Color', command=get_color).pack()
dialog.title('Simple Form')
dialog.mainloop()
```

Save the program into a file, called ch17_03.py. Then, run the program.

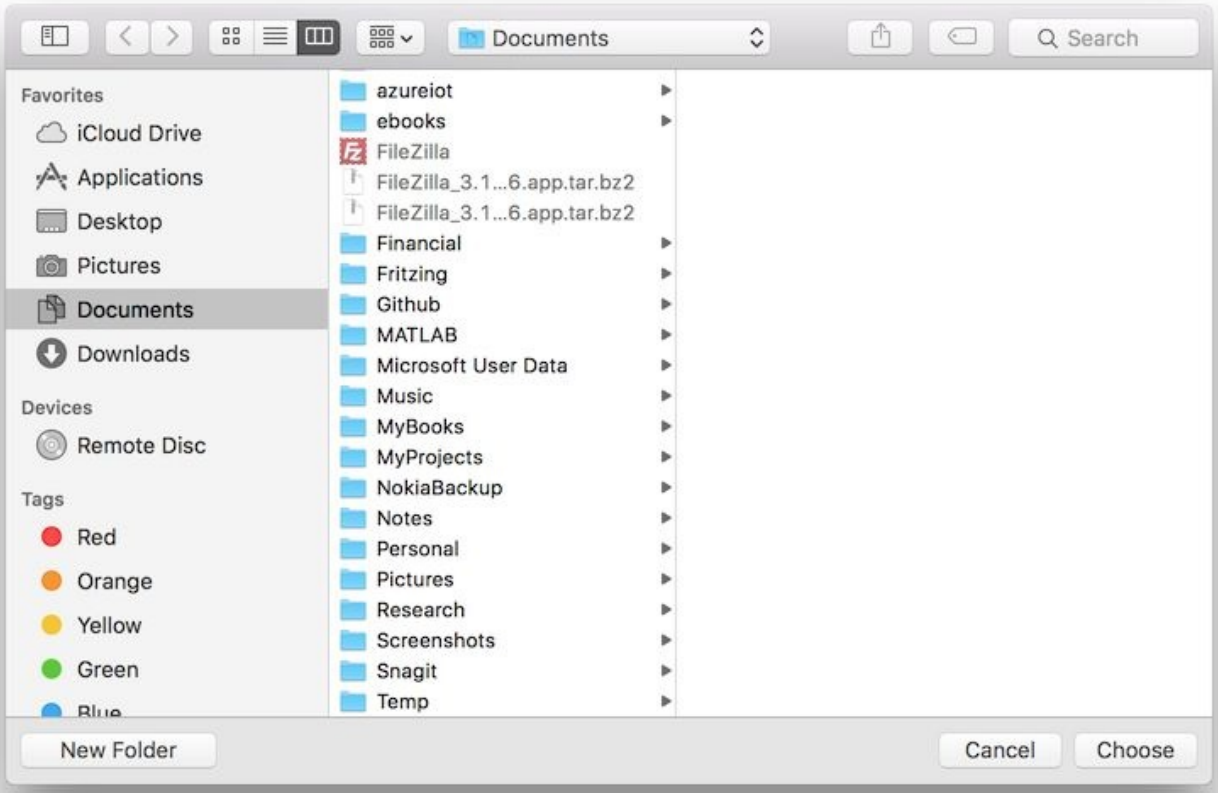
```
$ python3 ch17_03.py
```

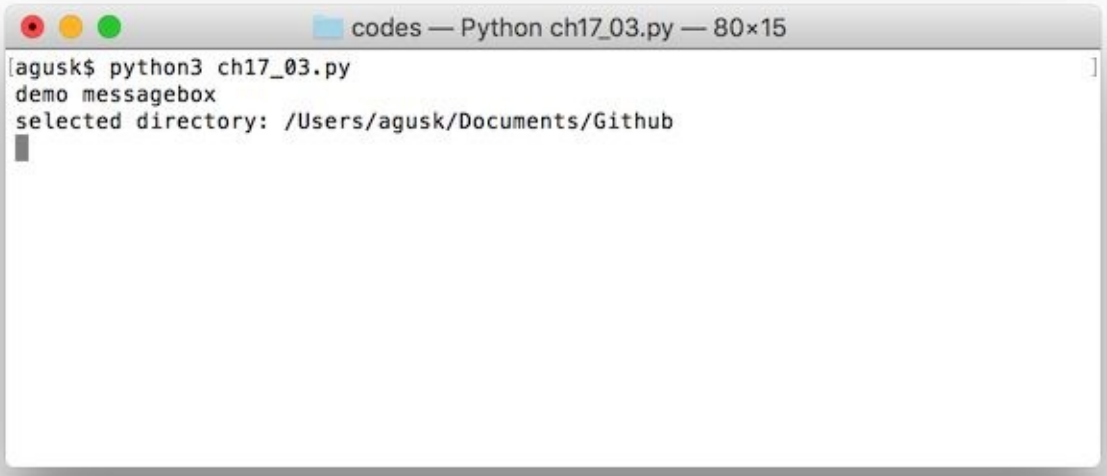
The following is output forms for information, error and warning.



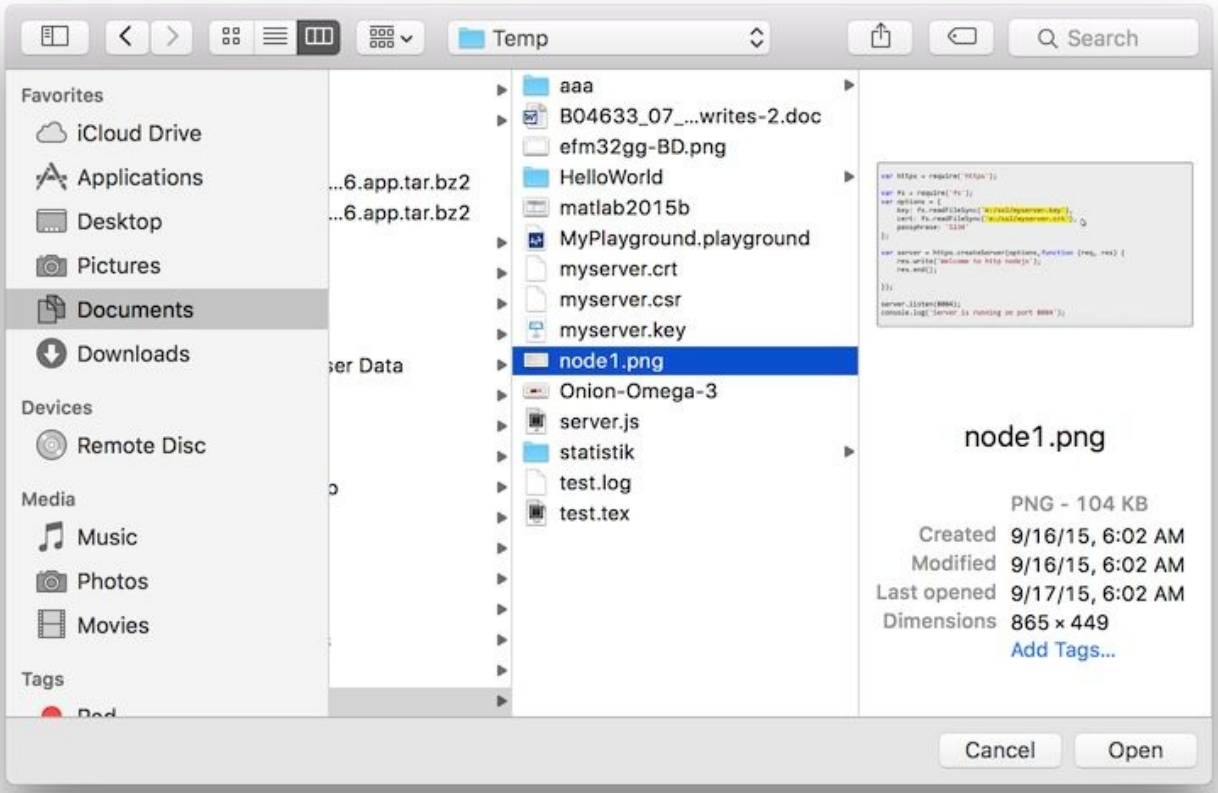


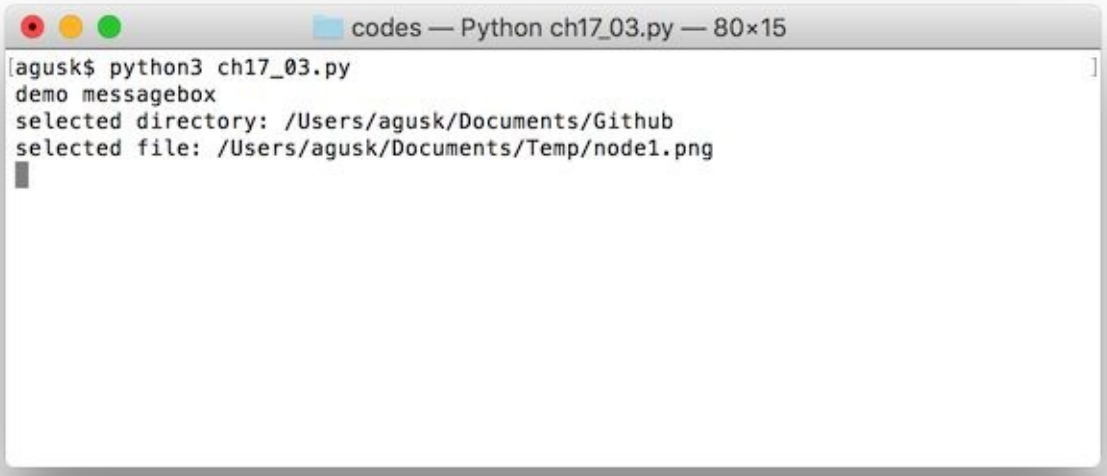
Output form for selecting a directory.



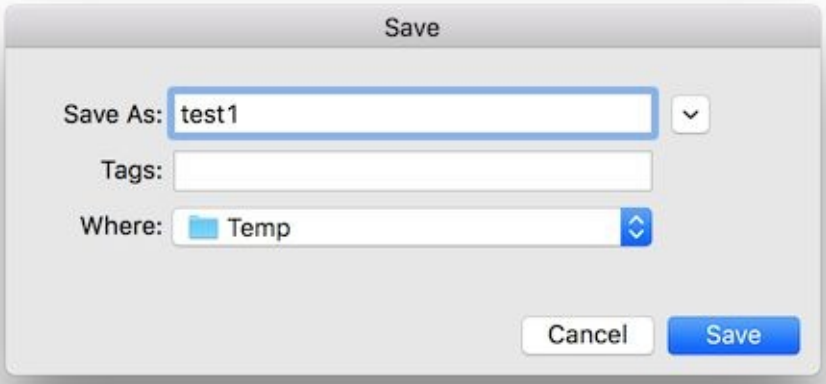


Output form for selecting a file.

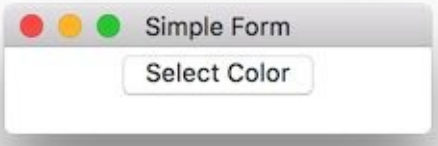




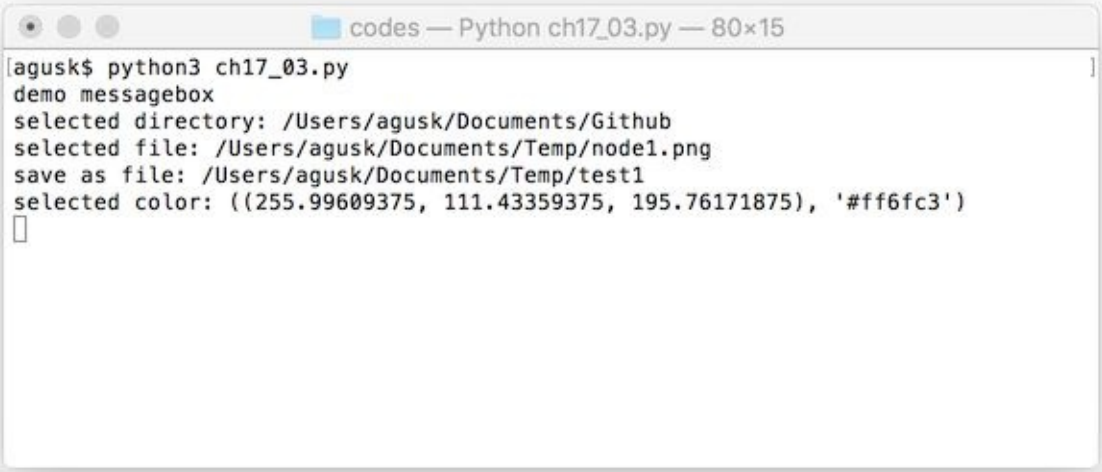
Output form for saving a file.



Output form for selecting a color. Click Select Color button to show Colors dialog.



The following is program output in Terminal.



18. Python Unit Testing

This chapter explains how to build unit testing in Python.

18.1 Getting Started

Unit testing is a software development process in which the smallest testable parts of an application, called units, are individually and independently scrutinized for proper operation. Unit testing can be used to minimize bugs on the application.

In Python, we can use unittest framework, https://docs.python.org/3/library/unittest.html . Please read it to obtain more information.

18.2 Demo

For testing, we create a class and do unit testing for this class.

Write a file, called mathu.py and write these scripts.

```
class Mathu:
    def __init__(self):
        print('call __init__ from Mathu class')

    def add(self, num_a, num_b):
        return num_a + num_b

    def div(self, num_a, num_b):
        try:
            result = num_a / num_b
        except ZeroDivisionError as e:
            raise e

        return result

    def check_even(self, number):
        return number % 2 == 0
```

Now we want to test this class by the following scenario:

- testing add()
- testin div()
- testing check_even()
- testing for exception error

You can write these scripts.

```
import unittest
import mathu


class TestMathu(unittest.TestCase):
    def test_add(self):
        res = mathu.Mathu().add(5, 8)
        self.assertEqual(res, 13)

    def test_div(self):
        res = mathu.Mathu().div(10, 8)
        self.assertGreater(res, 1)

    def test_check_even(self):
        res = mathu.Mathu().check_even(4)
        self.assertTrue(res)
```

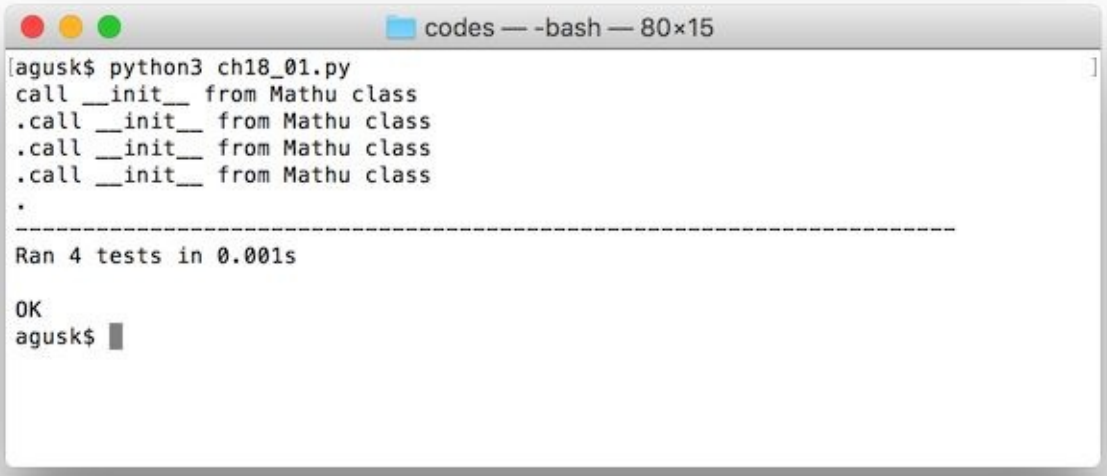
```
    def test_error(self):
        self.assertRaises(ZeroDivisionError, lambda:mathu.Mathu().div(5,

if __name__ == '__main__':
    unittest.main()
```

Save into a file, called ch18_01.py.

Now you can run the program.

```
$ python3 ch18_01.py
```

Program output:



Source Code

You can download source code on
http://www.aguskurniawan.net/book/python2120151.zip .

Contact

If you have question related to this book, please contact me at aguskur@hotmail.com . My blog: http://blog.aguskurniawan.net