# Programming Tutorial

## 04 Method

### Why Method?

1. Encapsulation of logics.
    - Codes which tells the same story should be encapsulated together.
2. Reuse.
    - Codes which will be used together in other place should be encapsulted together.
3. Easy to change.
    - Codes which will be changed together should be encapsulted together.

### Declare and Call a Method

```
#Declare
def say_hello():
    #block in a function
    print('Hello world')

#Call
say_hello()
```

### Parameter and Argument

1. Parameter
    - A parameter is the variable which is part of the method's signature (method declaration).
2. Argument
    - An argument is an expression used when calling the method.

```
def print_max(a,b):  #parameter a, b
    if a>b:
        print(a,'is maximum')
    elif a == b:
        print(a, 'is equal to',b)
    else:
        print(b,'is maximum' )

print_max(3,4)       # argument 3,4
```

```
x = 5
y = 7

print_max(x,y)       # argument x,y
```

Arguments and parameters are matched sequentially. a is set to 3. b is set to 4.

```
#!/usr/bin/python3

def printme( str ):
   print (str);
   return;

#call the method without argument
printme();
```

A error information will be shown like this:

> Traceback (most recent call last): File "test.py", line 10, in printme(); TypeError: printme() missing 1 required positional argument: 'str'

## Return Value

Method can utilize return value to send out the result of its calculation.

```
def sum( arg1, arg2 ):
   # 返回2个参数的和.
   total = arg1 + arg2
   print ("函数内 : ", total)
   return total;

# assign return value to a new variable
total = sum( 10, 20 );
print ("函数外 : ", total)

def myfun():
  return 1,2,3
a,b,c = myfun()
print(a,b,c)
```

## Main Fuction

The main fuction is the entrance for our code.

```
#hello.py
def foo():
    str="function"
    print(str);
if __name__=="__main__": ##程序的入口点
    print("main")
    foo()
```

## Local Variable

Each variable has its scope. Parameter is a local variable, which is valid only in its fuction.

```
x = 50
def func(x):  # x is parameter, a local variable
    print('x is', x)
    x = 2
    print('Changed local x to', x)

func(x)
print('x is still',x)
```

```
x = 50
def func():
    global x  # In python, you can use reserved word "global" to claim x is a
global variable
    print('x is', x)
    x = 2
    print('Changed local x to', x)

func()
print('x is still',x)
```

## Test your code

Don't submit any single code to code repository before you test your code.

Black-Box Testing is to do the testing without any konwledge about the code itself.

You only konw the function of the code, the inputs and their corresponding outputs.

The proceduce of block-box testing is shown below:

1. Initialization
2. Call the method to get the actual result
3. Compare the result with our expectation
4. Output the testing result

```
# Source code currency.py
def currency_exchange():
    euro = int(input("How many euros are you exchanging?"))
    rate = float((input("What is the exchange rate?")))
    dollar = round(euro*rate/100, 2)
    print("{0} euros at an exchange rate of {1} is {2} U.S.
dollars.".format(euro, rate, dollar))
```

```
# Test code currency_test.py
import unittest  # use unittest test framework
import currency  # import currency module
import sys


class TestCurrency(unittest.TestCase):
    """test currency exchange"""

    #Initialization
    def setUp(self):
        self.stream_out = MyStream()
        self.stream_in = MyStream()
        self.out_stream = sys.stdout
        self.in_stream = sys.stdin
        sys.stdout = self.stream_out
        sys.stdin = self.stream_in
        pass

    def test_currency1(self):
        euro = '97'
        rate = '31'
        self.stream_in.write(euro)
        self.stream_in.write(rate)
        #Call the method
        currency.currency_exchange()
        result = float(str(self.stream_out.buff[2]).split(" ")[9])
        #Compare the result with our expectation
        self.assertEqual(30.07,result)
        #Do not use assertTrue, which will omit the differece between
expectation and actual result.
        self.assertTrue(30.07 == result)
```