

Ch11: Debugging & Unit Testing

305172 Computer Programming
Laboratory
Jiraporn Pooksook
Naresuan University

Exercise

- Write a code to calculate the factorial of a non-negative number n .

Input: 5

output: 120

Input: 0

output: 1

Factorial Code

```
def fac(n, less_zero=False) :  
    if less_zero==True:  
        return 0  
    else:  
        ans=1  
        for i in range(1,n+1) :  
            ans=ans*i  
    return ans
```

Unit Test Code

```
import unittest
from factorial import fac

class TestFactorial(unittest.TestCase):

    def test_factorial(self):
        self.assertEqual(fac(5), 120)
        self.assertEqual(fac(-4, True), 0)

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

What is Unit Testing?

- *In computer programming, unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine if they are fit for use.*

Write a Unit Test in Python

Import a function that we want to test from the source file

```
import unittest
from factorial import fac
```

unittest library

```
class TestFactorial(unittest.TestCase):

    def test_factorial(self):
        self.assertEqual(fac(5), 120)
        self.assertEqual(fac(-4, True), 0)
```

Create a Test class extended from TestCase

assert statement

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

Run the test

Assert Statements

Method	Checks that
<code>assertEqual(a, b)</code>	<code>a == b</code>
<code>assertNotEqual(a, b)</code>	<code>a != b</code>
<code>assertTrue(x)</code>	<code>bool(x) is True</code>
<code>assertFalse(x)</code>	<code>bool(x) is False</code>
<code>assertIs(a, b)</code>	<code>a is b</code>
<code>assertIsNot(a, b)</code>	<code>a is not b</code>
<code>assertIsNone(x)</code>	<code>x is None</code>
<code>assertIsNotNone(x)</code>	<code>x is not None</code>
<code>assertIn(a, b)</code>	<code>a in b</code>
<code>assertNotIn(a, b)</code>	<code>a not in b</code>
<code>assertIsInstance(a, b)</code>	<code>isinstance(a, b)</code>
<code>assertNotIsInstance(a, b)</code>	<code>not isinstance(a, b)</code>

Exercise

- Write a code that receive a list of number as an input and then it returns a list of odd numbers as an output.
- Write a test code to test 2 examples:
 - Input: [1,2,3]
 - Output: [1,3]
 - Input: [2,4,6]
 - Output : None

Exercise

- Write a code that receive a list of number as an input and then it returns a list of odd numbers as an output.

```
def oddList(numList):  
    ans=[]  
    for i in numList:  
        if i%2==1:  
            ans.append(i)  
  
    if len(ans)==0:  
        return None  
    else:  
        return ans
```

Exercise

- Write a test code.

```
import unittest
from numberlist import oddList

class TestOddList(unittest.TestCase):

    def test_oddlist(self):
        self.assertEqual(oddList([1,2,3]), [1,3])
        self.assertIsNone(oddList([2,4,6]))

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

Exception Test

```
def div(x, y):  
    return x/y  
  
import unittest  
from division import div  
  
class TestDiv(unittest.TestCase):  
  
    def test_div(self):  
        self.assertRaises(ZeroDivisionError, div, 3, 2)  
        self.assertRaises(ZeroDivisionError, div, 3, 0)  
  
if __name__ == '__main__':  
    unittest.main()
```

Exceptions

Sr.No.	Exception Name & Description
1	Exception Base class for all exceptions
2	StopIteration Raised when the next() method of an iterator does not point to any object.
3	SystemExit Raised by the sys.exit() function.
4	StandardError Base class for all built-in exceptions except StopIteration and SystemExit.
5	ArithmeticError Base class for all errors that occur for numeric calculation.

- References: <https://docs.python.org/2/library/exceptions.html>
https://www.tutorialspoint.com/python/python_exceptions.htm

References

- <https://pymbook.readthedocs.io/en/latest/testing.html>
- <https://docs.python.org/2/library/unittest.html>

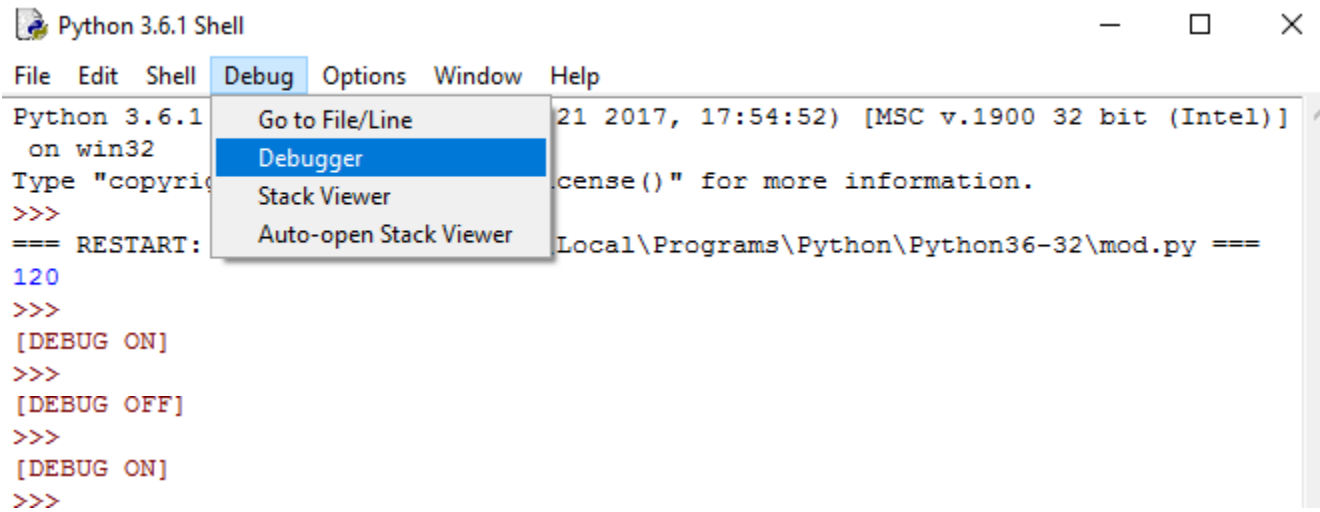
Exercise

- เขียนฟังก์ชันที่รับอินพุตเลข **10** จำนวนแล้ว ทำการรีเทิร์นค่าผลรวมของตัวเลขทั้ง **10** จำนวน
- เขียน **test code** เพื่อทดสอบฟังก์ชันนี้ด้วย
- ตัวอย่าง

Input: 5 1 2 9 -3 7 8 2 0 -4

Output: 27

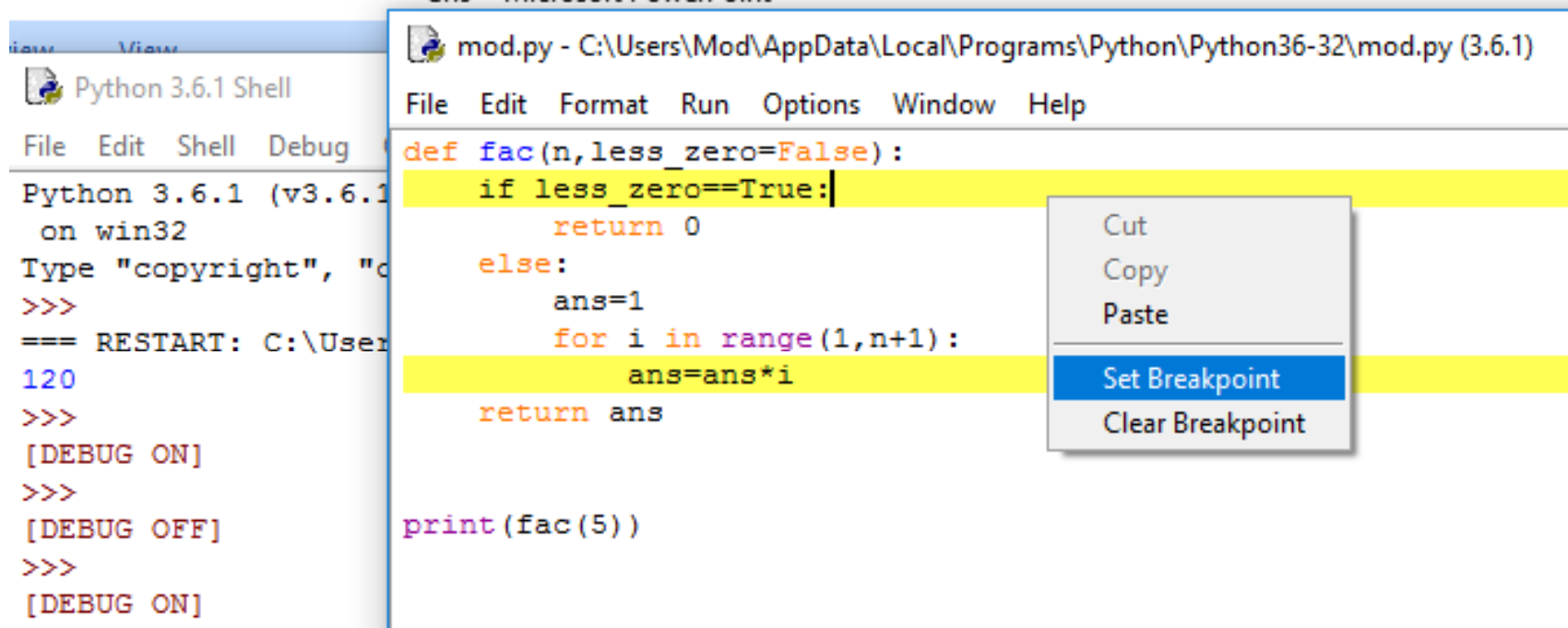
Debugging in Idle



The image shows a screenshot of the Python 3.6.1 Shell window. The window title is "Python 3.6.1 Shell" and it has standard Windows window controls (minimize, maximize, close). The menu bar includes "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The "Debug" menu is open, showing four options: "Go to File/Line", "Debugger" (which is highlighted in blue), "Stack Viewer", and "Auto-open Stack Viewer". The main text area of the shell contains the following text:

```
Python 3.6.1 Shell
Python 3.6.1
on win32
Type "copyright" for more information.
>>>
=== RESTART: Local\Programs\Python\Python36-32\mod.py ===
120
>>>
[DEBUG ON]
>>>
[DEBUG OFF]
>>>
[DEBUG ON]
>>>
```

Debugging in Idle



The screenshot shows the Python IDLE environment. On the left is the Python 3.6.1 Shell, and on the right is the code editor for a file named 'mod.py'. The code in the editor is a factorial function. A context menu is open over the code, with 'Set Breakpoint' highlighted.

```
mod.py - C:\Users\Mod\AppData\Local\Programs\Python\Python36-32\mod.py (3.6.1)
File Edit Format Run Options Window Help

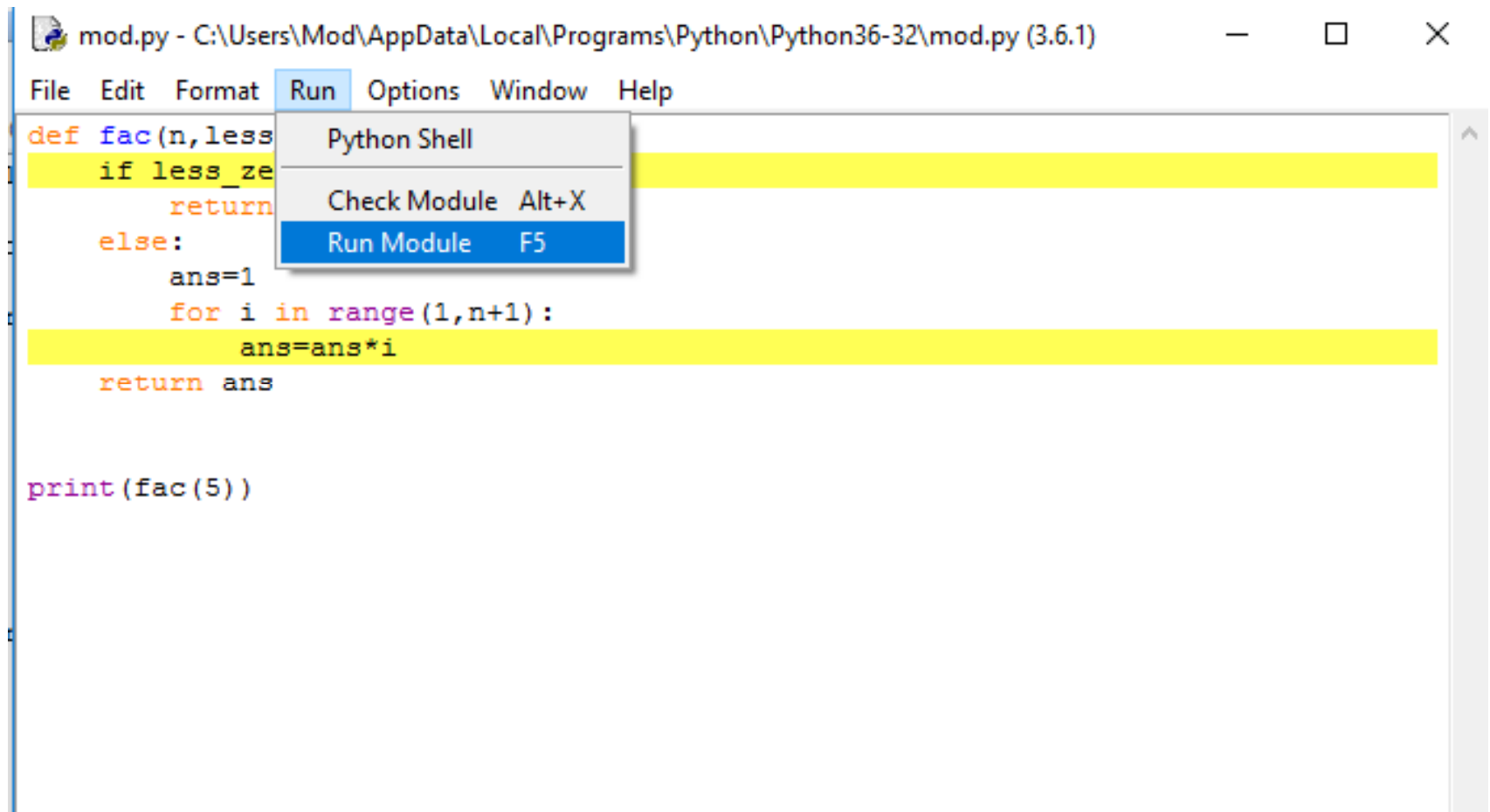
def fac(n, less_zero=False):
    if less_zero==True:
        return 0
    else:
        ans=1
        for i in range(1,n+1):
            ans=ans*i
        return ans

print(fac(5))
```

Python 3.6.1 Shell

```
Python 3.6.1 (v3.6.1) on win32
Type "copyright", "credits()" or "help()" to get more help.
>>>
=== RESTART: C:\Users\Mod\AppData\Local\Programs\Python\Python36-32\python.exe
120
>>>
[DEBUG ON]
>>>
[DEBUG OFF]
>>>
[DEBUG ON]
```


Debugging in Idle



The screenshot shows the Idle Python IDE window titled "mod.py - C:\Users\Mod\AppData\Local\Programs\Python\Python36-32\mod.py (3.6.1)". The menu bar includes "File", "Edit", "Format", "Run", "Options", "Window", and "Help". The "Run" menu is open, showing three options: "Python Shell", "Check Module Alt+X", and "Run Module F5". The "Run Module F5" option is highlighted in blue. The code in the editor is as follows:

```
def fac(n, less):
    if less_zero:
        return 1
    else:
        ans=1
        for i in range(1,n+1):
            ans=ans*i
        return ans

print(fac(5))
```

Debugging in Idle

The screenshot shows the Python Idle environment with a file named `mod.py` open. The code defines a factorial function `fac` and prints its result for `5`. The code is highlighted in yellow, indicating it is being debugged. The Debug Control window is open, showing the current execution state and local variables.

```
def fac(n, less_zero=False):
    if less_zero==True:
        return 0
    else:
        ans=1
        for i in range(1,n+1):
            ans=ans*i
        return ans

print(fac(5))
```

The Debug Control window shows the following information:

- Buttons: Go, Step, Over, Out, Quit
- Checkboxes: Stack, Source, Locals, Globals
- Current execution state: `mod.py:7: fac()`
- Stack trace:
 - 'bdb'.run(), line 431: exec(cmd, globals, locals)
 - '_main_'.<module>(), line 11: print(fac(5))
 - > '_main_'.fac(), line 7: ans=ans*i
- Locals window:

Variable	Value
ans	1
i	1
less_zero	False
n	5