Tutorial 8 Worksheet

COMP1117B Computer Programming I 2018-2019

|  |
| --- |
| **Tutorial objectives:**   * To help you understand and practice recursive principles in Python.   **Tutorial 8 Exercises Deadline: April 3, 2019 11:20** |

1. **Demo: The famous Fibonacci sequence**

The Fibonacci problem argues that, except for the first two elements in a list, each element equals the sum of the previous two items. It looks like:

[1, 1, 2, 3, 5, 8, 13, 21, ……]

where ‘2’ equals ‘1’+’1’, ‘3’equals ‘1’+’2’ …….

Write a program to show the first 20 items.

To understand this problem, we can easily define two variables to represent the previous two items. By using a ‘while’ loop, we can easily formalize this problem in code:

|  |
| --- |
| def Fibonacci():  a=0  b=1  while b < 1000:  print(b,end=',')  a, b = b, a+b  Fibonacci() |

Here ‘a, b = b, a + b’ means: The expression on the right will be executed before the assignment changes, that is, the right side is executed first, for example, the first loop gets b-->1, a+b --> 0+1 and then executes Assign a, b =1, 0+1, so after executing this, a=1, b=1.

The output for above example is:

[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,]

However, we can solve this problem in a recursive manner, by:

1. Set up a stop point (otherwise the recursive function will loop forever)

2. Call function recursively.

To see how it happens, a sample code is shown below.

|  |
| --- |
| def Fibonacci\_recursive():  list = []  for k in range(20):  if k == 0 or k == 1:  list.append(1)  else:  list.append(list[k-1]+list[k-2])  print(list)  Fibonacci\_recursive() |

Of course, you can count a sum by using recursive principle:

|  |
| --- |
| def func(k):  if k == 1 or k == 2:  return 1  else:  return func(k-1)+ func(k-2)  res = func(20)  print(res) |

On a whole, by using recursive function, you can simplify your code, making it easier to read and understand. Even more, you can get a more efficient program, which is absolutely important when you’re writing a huge project. Next, we will have three exercises to practice your skills of writing recursive functions.

**Exercise 8.1 Iterative multiplication**

Consider the following program.

|  |
| --- |
| def iterative\_factorial(n):  result = 1  for i in range(2,n+1):  result \*= i  return result  res = iterative\_factorial(5)  print(res) |

This program will output the product of the first five positive integers. Now, please write a program to output the product of the first N positive integers by taking the input N.

|  |  |  |
| --- | --- | --- |
| **Test case** | **Input** | **Output** |
| Case 1 | 5 | 120 |
| Case 2 | 6 | 720 |

|  |  |  |
| --- | --- | --- |
| Case 3 | 10 | 3628800 |

|  |  |  |
| --- | --- | --- |
| Case 4 | 20 | 2432902008176640000 |

**Exercise 8.2 Power computation**

Write a Python program to calculate the value of 'a' to the power 'b'.

For example: power(3,4) -> 81

Take the input a and b, you are required to output the value of 'a' to the power 'b' in a recursive manner.

|  |  |  |
| --- | --- | --- |
| **Test case** | **Input** | **Output** |
| Case 1 | 3  4 | 81 |
| Case 2 | 2  9 | 512 |

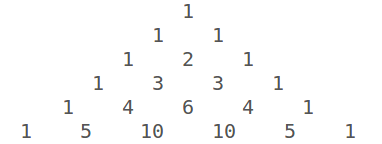
|  |  |  |
| --- | --- | --- |
| Case 3 | 4  5 | 1024 |

|  |  |  |
| --- | --- | --- |
| Case 4 | 1  0 | 1 |

|  |  |  |
| --- | --- | --- |
| Case 5 | 0  1 | 0 |

**Exercise 8.3 The Pascal's triangle**

Consider an interesting triangle which looks like:



The above triangle has several interesting principles:

1. In each line, both the first and the last integer are ‘1’.

2. Except for the first two lines in this triangle, each item in a line equals the sum of its above two integers.

For instance, in line 3: ‘1 2 1’, the integer ‘2’ equals the sum of its above two integers: ‘1’ and ‘1’.

Now, you are required to write a recursive program to simulate the Pascal’s triangle. Take the input N, you are required to output a list in line N. Please note that in this question, the index of a line starts from ‘1’ rather than ‘0’

|  |  |  |
| --- | --- | --- |
| **Test case** | **Input** | **Output** |
| Case 1 | 3 | [1, 2, 1] |
| Case 2 | 6 | [1, 5, 10, 10, 5, 1] |

|  |  |  |
| --- | --- | --- |
| Case 3 | 10 | [1, 9, 36, 84, 126, 126, 84, 36, 9, 1] |

|  |  |  |
| --- | --- | --- |
| Case 4 | 13 | [1, 12, 66, 220, 495, 792, 924, 792, 495, 220, 66, 12, 1] |