COMPSCI 1©1

Principles of Programming

Lecture 9 – Divide a problem into different tasks and define functions which perform each task, trace the execution of a small program which contains simple functions

Learning outcomes

At the end of this lecture, students should be able to:

- break a program into small tasks which can be implemented using functions
- know how to trace code which involves functions

From lecture 8

Recap

- write functions which perform a well defined task
- understand that a function can call other functions
- understand the scope of variable
- always use descriptive function names (and variable names) to ensure that the purpose of the function is clear

```
def get_discount(amount, rate):
    #Code not shown here
def get discount message(discount, rate):
    #Code not shown here
def print docket(price, discount rate):
    discount amt = get discount(price, discount rate)
    discount message = get discount message(discount amt,
                                                discount rate)
    print("Original price $" + str(price))
    print(discount_message)
                                            Original price $234
    final price = cost - discount amt
                                             5% Discount: $11.7
    print("Price $" + str(final price))
                                            Price $222.3
print docket(234, 5)
                                             Original price $657
print()
                                             15% Discount: $98.55
print_docket(657, 15)
                                             Price $558.45
```

Madlibs

A madlib is the name for a simple game. The idea is to take a sentence and remove some words. You then ask someone to enter some words which fit the same general category as the removed words and see the new sentence which is created:

[Mary] had a little [lamb], its fleece was [white] as [snow]. Everywhere that [Mary] went, the [lamb] was sure to [go].

[NAME] had a little [ANIMAL], its fleece was [COLOUR] as [PLURAL_NOUN] Everywhere that [NAME] went, the [ANIMAL] was sure to [ACTION]

Think about the functions needed to write this program (2 functions) and write the carry_out_madlib() function code for this program.

Madlibs

```
Complete the
def get word(prompt):
                                                    carry_out_madlib()
  word = input("Enter " + prompt + ": ")
                                                         function.
  return word
def display madlib(name, animal, colour, compare thing, go word):
  stars = "*" * 35
  print(stars)
  print(name + " had a little " + animal + ",")
  print("its fleece was " + colour + " as " + compare thing + ".")
  print("Everywhere that " + name + " went,")
  print("the " + animal + " was sure to " + go word + ".")
  print(stars)
def carry out madlib():
  prompt name = "a name"
  prompt animal = "an animal"
  prompt colour = "a colour"
  prompt_thing = "a plural noun (thing)"
  prompt action = "an action"
                              #Complete the code in this function
carry out madlib()
```

Format of CompSci 101 programs from here on

```
1
   def function1(...):
2
        print("Executing function1()")
3
        ....
4
   def function2(...):
5
        print("Executing function2()")
6
        ....
7
   ...
   def main():
8
        print("Executing main()")
9
        \dots = function1(\dots)
10
        \dots = function2(\dots)
11
12 main()
```

Code trace – the program stack

1	<pre>def fun_2(age):</pre>
2	years = age + 10
3	<pre>print("3.", years)</pre>
4	<pre>def fun_1(years):</pre>
5	<pre>print("4.", years)</pre>
6	years = 20
7	<pre>def main():</pre>
8	years = 5
9	<pre>fun_1(years)</pre>
10	<pre>print("1.", years)</pre>
11	<pre>fun_2(years)</pre>
12	<pre>print("2.", years)</pre>
10	main()

13

main(

This program starts executing on the first unindented line of code (line 13).

Every time a function is called (lines 13, 9 and 11), a section of space in the program memory is set aside for the parameters and the local variables of the called function.

When the function finishes executing, the space set aside for the function is freed (released).

Code trace – the program stack

5

5

5

15

4.

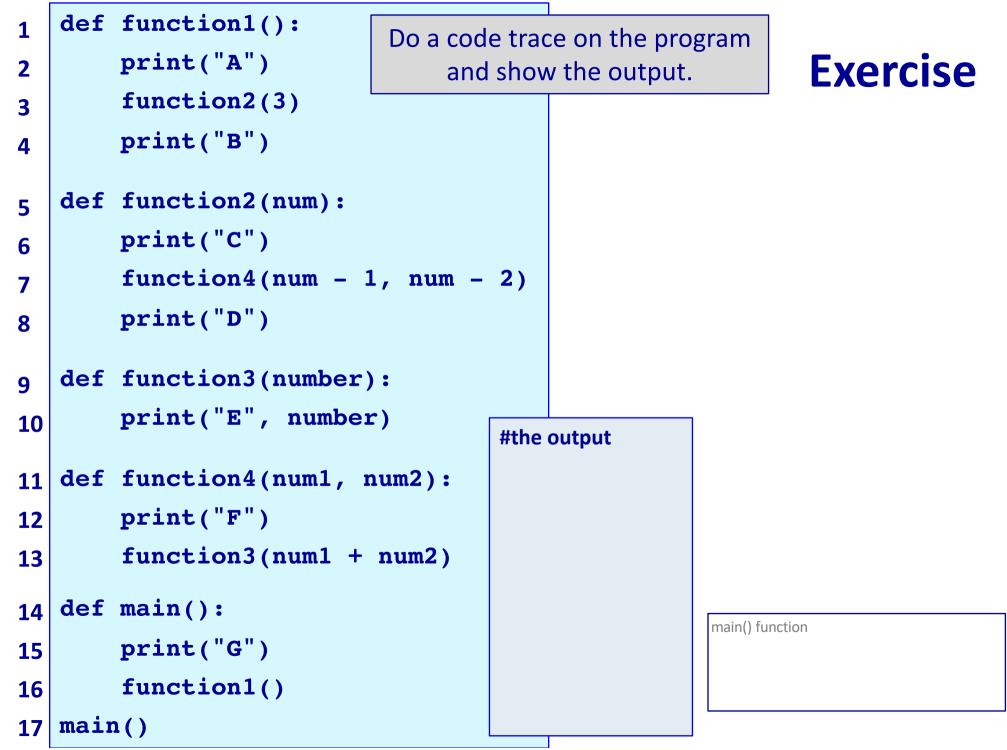
1.

3.

2.

1	<pre>def fun_2(age):</pre>		
2	years = age + 10		
3	<pre>print("3.", years)</pre>		
4	<pre>def fun_1(years):</pre>		
5	<pre>print("4.", years)</pre>		
6	years = 20		
7	<pre>def main():</pre>		
8	years = 5		
9	<pre>fun_1(years)</pre>		
10	<pre>print("1.", years)</pre>		
11	<pre>fun_2(years)</pre>		
12	<pre>print("2.", years)</pre>		
13	<pre>main()</pre>		

fun_2() function		
age 5		
years 15		
fun_1() function		
years 5	20	
main() function		
years 5		



Code trace example

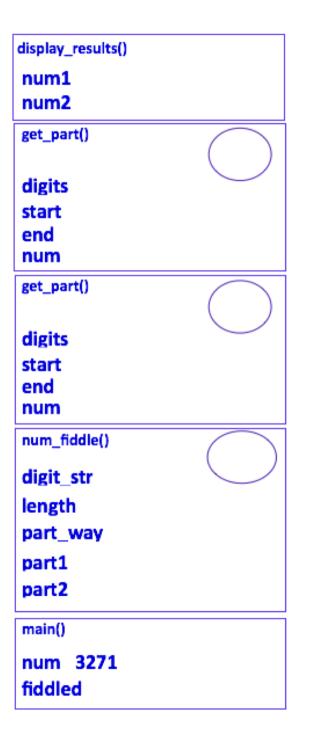
def get part(digits, start, end):

1

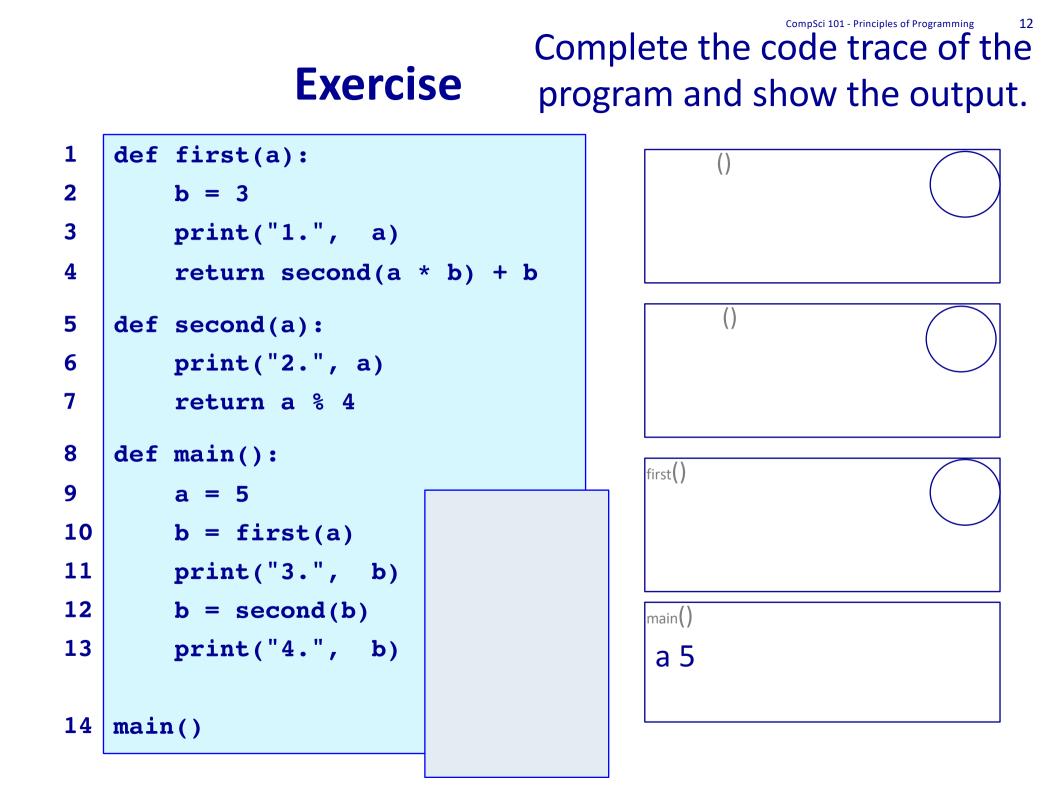
Part of the code trace for this program is shown on the next slide. The rest of the code trace will be shown in lectures.

```
2
       num = int(digits[start: end])
3
       return num
   def num fiddle(digit str, length):
4
       part way = length // 2
5
       part1 = get_part(digit_str, 0, part_way)
6
       part2 = get part(digit str, part way, length)
7
       return part1 + part2
8
   def display results(num1, num2):
9
       print(num1, ", ", num2, sep = "")
10
   def main():
11
12
       num = 3271
13
       fiddled = num fiddle(str(num), len(str(num)))
       display results(num - 5, fiddled)
14
15
   main()
```

Code Trace of the program on Slide 10



This code trace example will be finished in lectures.



Summary

Problems can be broken down into small tasks and each small tasks can be implemented using a function

A code tracing technique is used to work through the execution of a program, instruction by instruction.