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, you will know that:

- a program should be broken up into small tasks which can be implemented using functions

At the end of this lecture, you will understand how:

- to trace code which involves functions

Recap

From lecture 8

- 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)
    final_price = cost - discount_amt
    print("Price $" + str(final_price))

print_docket(234, 5)
print()
print_docket(657, 15)
```

```
Original price $234
5% Discount: $11.7
Price $222.3

Original price $657
15% Discount: $98.55
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 carry_out_madlib() function.

```
def get_word(prompt):
    word = input("Enter " + prompt + ": ")
    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

```
def function1(…):
    print("Executing function1()")
    ….

def function2(…):
    print("Executing function2()")
    ….

…

def main():
    print("Executing main()")
    … = function1(…)
    … = function2(…)

main()
```

Code trace – the program stack

```
1   def fun_2(age):
2       years = age + 10
3       print("3.", years)

4   def fun_1(years):
5       print("4.", years)
6       years = 20

7   def main():
8       years = 5
9       fun_1(years)
10      print("1.", years)
11      fun_2(years)
12      print("2.", years)

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.

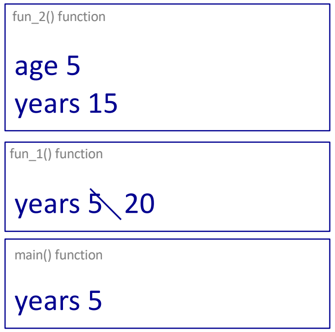
When the function finishes executing, the space set aside for the function is freed (released).

This code tracing technique will be shown in lectures.

Code trace – the program stack

```
1   def fun_2(age):
2       years = age + 10
3       print("3.", years)

4   def fun_1(years):
5       print("4.", years)
6       years = 20

7   def main():
8       years = 5
9       fun_1(years)
10      print("1.", years)
11      fun_2(years)
12      print("2.", years)

13  main()
```

```
4. 5
1. 5
3. 15
2. 5
```



fun_2() function

age 5
years 15

fun_1() function

years ~~5~~ 20

main() function

years 5

Exercise

Do a code trace on the program and show the output.

```
def function1():
    print("A")
    function2(3)
    print("B")

def function2(num):
    print("C")
    function4(num – 1, num – 2)
    print("D")

def function3(number):
    print("E", number)

def function4(num1, num2):
    print("F")
    function3(num1 + num2)

def main():
    print("G")
    function1()
main()
```

#the output

main() function

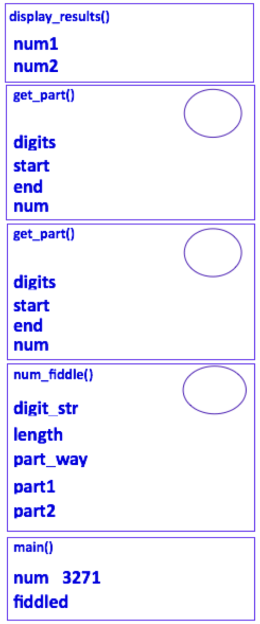
Code trace example

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.

```
def get_part(digits, start, end):
    num = int(digits[start: end])
    return num

def num_fiddle(digit_str, length):
    part_way = length // 2
    part1 = get_part(digit_str, 0, part_way)
    part2 = get_part(digit_str, part_way, length)
    return part1 + part2

def display_results(num1, num2):
    print(num1, ", ", num2, sep = "")

def main():
    num = 3271
    fiddled = num_fiddle(str(num), len(str(num)))
    display_results(num – 5, fiddled)

main()
```

Code Trace of the program on Slide 10

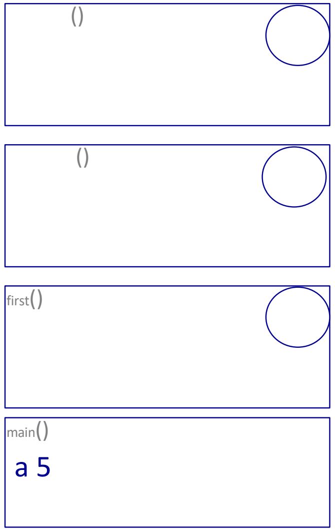
This code trace example will be finished in lectures.



Exercise

Complete the code trace of the program and show the output.

```
def first(a):
    b = 3
    print("1.",  a)
    return second(a * b) + b

def second(a):
    print("2.", a)
    return a % 4

def main():
    a = 5
    b = first(a)
    print("3.",  b)
    b = second(b)
    print("4.",  b)

main()
```



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.