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
Recap

- From lecture 8
  - write functions which perform a task
  - understand that a function can call another function
  - understand the scope of variable
  - always use excellent function names and variable names to ensure that the purpose of the function is clear

```python
def get_discount(amount, rate):
    discount = amount * rate / 100
    return round(discount, 2)

def get_discount_message(discount, rate):
    rate_message = str(rate) + "%"
    message = rate_message + " Discount: $" + str(discount)
    return message

def print_docket(cost, discount_rate):
    #Code not shown here

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

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

Original price $657
15% Discount: $98.55
Price $558.45
Wait for the user to press Enter (return)

- Sometimes we want the user to press Enter when they are ready:

Think of a number (press enter to continue):
Add 5 to the number (press enter to continue):
Times the number by 4 (press enter to continue):
Subtract 8 (press enter to continue):
Times the number by 2 (press enter to continue):
Divide the number by 8 (press enter to continue):
Subtract your original number (press enter to continue):

Any number such as 408
Add 5: 413
Times 4: 1652
Subtract 8: 1644
Times by 2: 3288
Divide by 8: 411
Subtract original number: 3

================================
The number you now have is 3
================================
Wait for the user to press Enter (return)

- Code for the previous slide.

```python
def prompt_continue(prompt):
    ... #code is left out here

def display_details():
    number = random.randrange(0, 500)
    ... #code is left out here
    display_final_number(number)

def display_final_number(number):
    ...
    message = " (press enter to continue)"
    prompt_continue("Think of a number" + message)
    prompt_continue("Add 5 to the number" + message)
    prompt_continue("Times the number by 4" + message)
    prompt_continue("Subtract 8 (press enter to continue)"")
    prompt_continue("Times the number by 2" + message)
    prompt_continue("Divide the number by 8" + message)
    prompt_continue("Subtract your original number" + message)
    display_details()
```

CompSci 101 - Principles of Programming
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 main() function code for this program.
Madlibs

- From the previous slide.

#write the main code below
Format of CompSci 101 programs from here on

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

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

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

def main():
    years = 5
    fun_1(years)
    print("1.", years)
    fun_2(years)
    print("2.", years)
    main()
```

---

**fun_2() function**

- age 5
- years 15

**fun_1() function**

- years 5
- 20

**main() function**

- years 5
- 5
- 1
- 5
- 3
- 15
- 2
- 5
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()
def get_part(digits, start, end):
    num = int(digits[start: end])
    return num

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

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

def main():
    num = 3271
    fiddled = number_fiddle(str(num), len(str(num)))
    display_results(num - 5, fiddled)
main()
This code trace example will be finished in lectures.
Complete the code trace of the program and show the output.

```python
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()
```

1. `def first(a):`
2. `b = 3`
3. `print("1", a)`
4. `return second(a * b) + b`
5. `def second(a):`
6. `print("2", a)`
7. `return a % 4`
8. `def main():`
9. `a = 5`
10. `b = first(a)`
11. `print("3", b)`
12. `b = second(b)`
13. `print("4", b)`
14. `main()`

Output:
```
1 5
2 15
3 6
2 6
4 2
```
Summary

- Problems can be broken down into small tasks and the small tasks can be implemented using functions

- A code tracing technique is used to show the execution of a program