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
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 variables inside the function
  - 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
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 use the main() function code for this program.
Madlibs

- A simple implementation.

```python
def show_madlib(name, animal, colour, compare, go_word):
    stars = "*" * 35
    print();
    print(stars)
    print(name + " had a little " + animal + ",")
    print("its fleece was " + colour + " as " + compare + ".")
    print("Everywhere that " + name + " went,"")
    print("the " + animal + " was sure to " + go_word + ".")
    print(stars)
    print()
```
Madlibs

- From the previous slide.

```python
mary = input("Enter a name: ")
lamb = input("Enter an animal: ")
white = input("Enter a colour: ")
snow = input("Enter a plural noun (thing): ")
go_word = input("Enter an action: ")
show_madlib(mary, lamb, white, snow, go_word)
```

- The main function approach:

```python
def main():
    mary = get_word("Enter a name")
    lamb = get_word("Enter an animal")
    white = get_word("Enter a colour")
    snow = get_word("Enter a plural noun (thing)"
    go_word = get_word("Enter an action")
    show_madlib(mary, lamb, white, snow, go_word)
main()
```
def function1(...):
    print("Executing function1()")
    ...

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

...

def main():
    function1(...)
    print("Executing main()")
    function2(...)

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).
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()
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()
display_results() function
num1
num2

get_part() function

digits
start
end
num

get_part() function

digits
start
end
num

number_fiddle()

digit_str
number_of_digits
part_way
part1
part2

main() function
num 3271
fiddled
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.