Lecture 8 – More practice defining functions, functions can call other functions, the scope of variables
At the end of this lecture, students should be able to:

- 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
Recap

- From lecture 7
  - functions which accept arguments and return values can be defined
  - calls to functions which have been defined are made (below the function definition)
  - we must use excellent function names and variable names to ensure that the purpose of the function is clear
  - each function performs one task

```python
def get_dice_total():
    dice1 = random.randrange(1, 7)
    dice2 = random.randrange(1, 7)
    return dice1 + dice2

def get_discount_price(price):
    discount_price = price * 0.95
    return discount_price

dice_throw = get_dice_total()
final_price = get_discount_price(345.67)
```
Syntax of a Python function

- A Python function has the following syntax:

```python
def function_name(comma_separated_arguments):
    statements in the function
    return value_to_be_returned
```

- `def` statements are followed by a colon and start the function.
- Arguments are passed in the parentheses.
- Statements in the function body can include any Python code.
- Use either 1 tab or 4 spaces for indentation.
- The `return` statement ends the function and returns a value.
Functions with no return statement

If a function does not need to return a result, then an optional return statement can be used as the last statement of the function (see lines 4 and 8), e.g.,

```python
def display_welcome(name):
    message = "Welcome **" + name + " **"
    print(message)
    return

def display_cost(dollars, cents):
    cost_str = "Cost is $" + str(dollars) + ":" + str(cents)
    print(cost_str)
    return

display_welcome("Sam")
print()
display_cost(15, 35)
```

Welcome **Sam**

Cost is $15:35
Functions with no return statement

- If a function does not need to return a result, then the last statement (the return statement) can be omitted. The following program is exactly the same as the program on the previous slide.

```python
def display_welcome(name):
    message = "Welcome **" + name + " **"
    print(message)

def display_cost(dollars, cents):
    cost_str = "Cost is $" + str(dollars) + ":" + str(cents)
    print(cost_str)

display_welcome("Sam")
print()
display_cost(15, 35)
```

Welcome **Sam**

Cost is $15:35
Functions with no return statement

- In Python, functions which do not explicitly return anything, return the value **None** by default.

```python
def display_welcome(name):
    message = "Welcome **" + name + " **"
    print(message)

def display_cost(dollars, cents):
    cost_str = "Cost is $" + str(dollars) + ":" + str(cents)
    print(cost_str)

print(display_welcome("Sam"))
print()
result = display_cost(15, 35)
print(result)
```

See slide 14 of lecture 4:
**None** is a special value which can be assigned to a variable and it means that the variable is not referencing (pointing to) any object.
Functions with no parameters

- Functions may not need to have any parameters inside the round brackets. If the function does not need to receive any information in order to do its job then there will not be any parameters in its parameter list.

```python
1 def display_intro():
2     message = "Game of Nim"
3     print(message)

4 def display_menu():
5     print("1. Option 1")
6     print("2. Option 2")
7     print("3. exit")

8 display_intro()
9 print()
10 display_menu()
```
Python - indentation

- Python programs are structured through indentation
  - All programming languages use blocks of code and in all programming languages, it is desirable that blocks of code be indented (a style requirement, not a language requirement). This principle makes it easier to read and understand other people's code.

- In Python indentation of blocks of code is a language requirement not a matter of style. All statements with the same distance from the left belong to the same block of code, i.e., the statements within a block line up vertically. The block ends at a less indented line or the end of the file. If a block has to be more deeply nested, it is simply indented further to the right.

```python
import blah
n = blahblahblah
n = n + blahblahblah
blahblahblahblahblahblah:
    blahblahblah:
        c1 = blahblahblah
c2 = blahblahblah
blahblahblahblahblahblah:
    blahblahblah
    blahblahblah
print("The end")
```
Python code is structured through indentation (the skeleton of a program below contains no functions):

- Block 1
  - Block 2
    - Block 3
    - Block 2 continued
  - Block 1 continued
Python code is structured through indentation (example skeleton contains two functions):

- Defining functions: `def ...`
- Function body:
  
- Blocks:
  - Block 1
  - Block 2 (continued)
  - Block 3

Maybe some import statements here

Two function definitions at the top of the program
The use of colons (:) is another aspect of Python program structure

- The statement marking the beginning of an indented block ends with a colon.

```python
def display_intro():
    message = "Game of Nim"
    print(message)

def display_winner_details(winner, score):
    message = "*** " + winner.upper() + " (" + str(score) + ") ***"
    print(message)

display_intro()
print()
display_winner_details("Jo Li", 56)
```

Game of Nim
*** JO LI (56) ***
Each statement marking the beginning of an indented block ends with a colon (example program diagram contains no functions):
Each statement marking the beginning of an indented block ends with a colon (example program diagram contains two functions):
A Python program starts executing at the first unindented statement (line 7 in the code below).

When the Python interpreter comes across statements (other than def or import ... and a few other keywords) which are written in the left-most column, it will start the program by executing these statements.

def display_intro():
    message = "Game of Nim"
    print(message)

def display_winner_details(winner, score):
    message = "*** " + winner.upper() + " (" + str(score) + ") ***"
    print(message)

display_intro()
display_winner_details("Jo Li", 56)

Game of Nim
*** JO LI (56) ***
The following program will execute without error but there is no output.

```python
def display_intro():
    message = "Game of Nim"
    print(message)

def display_winner_details(winner, score):
    message = "*** " + winner.upper() + " (" + str(score) + ")
    print(message)
```

The code in the two functions is executed. You can verify this: put an error into one part of the function code (e.g., put `print(message)` in line 6) and you will see that the interpreter will display the error.
Local variables

- Local variables and their scope
  - When you set the value of a variable inside a function, the Python interpreter creates a **local variable** with that name.
  - In the following example, the variables: `message`, `author`, `length` and `symbols` are local variables defined inside the `display_intro()` function.
  - In a function, local variables exist from the moment they are set (used) to the end of the function block inside which they are used. For example the variable, `author`, exists from line 3 to line 9.

```python
1 def display_intro():
2     message = "Game of Nim"
3     author = "by Adriana Ferraro"
4     length = max(len(message), len(author))
5     symbols = "*" * length
6     print(symbols)
7     print(message)
8     print(author)
9     print(symbols)
10    display_intro()
```

Game of Nim
by Adriana Ferraro

***************
Variables – out of scope

When you try to use a variable which is **out of scope**, the interpreter will give an error:

```python
def display_intro():
    message = "Game of Nim"
    author = "by Adriana Ferraro"
    length = max(len(message), len(author))
    symbols = "*" * length
    print(symbols)
    print(message)
    print(author)
    print(symbols)

display_intro()
print(author)
```

```
***************
Game of Nim
by Adriana Ferraro
***************
Traceback (most recent call last):
  File "OutOfScopeExample.py", line 11, in <module>
    print(author)
NameError: name 'author' is not defined
```
Exercise

- Complete the output from the following program.

```python
1 def display_intro():
2     message = "Game of Nim by Adriana Ferraro"
3     length = len(message)
4     symbols = "*" * length
5     print(symbols)
6     print(message)
7     print(symbols)
8     message = "bye bye!"
9     display_intro()
10    print(message)
```

Game of Nim by Adriana Ferraro

****************************
Game of Nim by Adriana Ferraro
****************************
The scope of parameters

- Parameters are the variables which are listed in the function header.
  - The **scope of parameters** is the same as for local variables, i.e., they exist from the moment they are set (at the beginning of the function) to the end of the function block inside in which they are listed, i.e., the end of the function definition. In the example below the parameters, `winner` and `score`, exist from line 1 to line 4.

```python
1 def display_winner_details(winner, score):
2     message = "*** " + winner.upper() + " (" +
3     message = message + str(score) + ") ***"
4     print(message)

5 display_winner_details("Jo Li", 56)
```

*** JO LI (56) ***
Example with four function calls

```
1 def get_winner_message(name):
2     message = "*** Game of Nim: " + name + " is the winner ***"
3     return message
4
def display_winner_details(score, winner_message):
5     message = "(" + str(score) + " points)"
6     number_of_blanks = (len(winner_message) - \
                          len(message)) // 2
7     blanks = " " * number_of_blanks
8     print(winner_message)
9     print(blanks + message)

10 message = get_winner_message("Sam")
11 display_winner_details(66, message)
12 message = get_winner_message("Helen")
13 display_winner_details(178, message)
```

*** Game of Nim: Sam is the winner ***
(66 points)

*** Game of Nim: Helen is the winner ***
(178 points)
A function can call another function

```python
def get_winner_message(name):
    message = "*** Game of Nim: " + name + " is the winner ***"
    return message

def display_winner_details(winner, score):
    score_message = "(" + str(score) + " points)"
    winner_message = get_winner_message(winner)
    number_of_blanks = (len(winner_message) - \
                        len(score_message)) // 2
    blanks = " " * number_of_blanks
    print(winner_message)
    print(blanks + score_message)

display_winner_details("Sam", 66)
display_winner_details("Helen", 178)
```

*** Game of Nim: Sam is the winner ***
(66 points)

*** Game of Nim: Helen is the winner ***
(178 points)

This program does exactly the same job as the program on the previous slide.
Complete the get_discount() function which returns the discount amount (a float number rounded to 2 decimal places). The function is passed two arguments, the amount and the discount rate.

```python
def get_discount(amount, rate):
    discount_message = "Discount: $" + str(get_discount(234, 5))
    print(discount_message)
    discount_message = "Discount: $" + str(get_discount(125, 15))
    print(discount_message)
```

Discount: $11.7
Discount: $18.75
Exercise

- Complete the `get_discount_message()` function which returns a string made up of the rate of discount, followed by the string "% Discount: ", followed by the discount amount. The function has two parameters, the discount amount and the discount rate.

```python
def get_discount_message(discount, rate):
    message = get_discount_message(11.7, 5)
    print(message)
    message = get_discount_message(98.55, 15)
    print(message)
```

5% Discount: $11.7
15% Discount: $98.55
Exercise

- Complete the `print_docket()` function which prints the sales docket information (the format should be as shown in the output shown). The function is passed two arguments, the cost price and the discount rate. Your function code MUST make a call to both the functions: `get_discount()`, `get_discount_message()`.

```python
def get_discount(amount, rate):
    # code from slide 23

def get_discount_message(discount, rate):
    # code from slide 24

def print_docket(cost, discount_rate):

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

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

Original price $657
15% Discount: $98.55
Price $558.45
Exercise

- The following program prompts the user for a number of items to be packaged. Each box can hold 10 items. Any left over items require an extra box. The first 6 boxes cost $8 each and any boxes above the first 6, cost $5 each. The program executes as shown in the example outputs below. **Design the functions needed** to write this program and write the main code for the program.

```
Enter number of items: 20
Items: 20
Boxes needed: 2
Cost: $16
```
```
Enter number of items: 36
Items: 36
Boxes needed: 4
Cost: $32
```
```
Enter number of items: 65
Items: 65
Boxes needed: 7
Cost: $53
```
```
Enter number of items: 102
Items: 102
Boxes needed: 11
Cost: $73
```
Exercise

- From the previous slide.

```python
#write the main code below
items_per_box = 10

Enter number of items: 102
Items: 102
Boxes needed: 11
Cost: $73
```
The following program has two errors. What are the errors? Write a correction for each error.

The desired output is shown below the program.

```python
1 def display_winner_details(winner, score):
2     message = "*** " + winner.upper() + " ("
3     message = message + score + ") ***"
4     print(message)
5     score = score + 50
6 display_winner_details("Jo Li", score)
7 print(score)
```

*** JO LI (50) ***
50
In a Python program:

- functions can be used to perform various tasks
- a function can make a call to another function
- the scope of variable needs to be understood
- It is important to always use excellent function names and variable names to ensure that the purpose of the function is clear
Examples of Python features used in this lecture

```python
def display_welcome(name):
    message = "Welcome **" + name + " **"
    print(message)
    return

def display_intro(name):
    local_variable = "Game of Nim"
    local_variable = local_variable + "by " + name
    print(local_variable)

def display_menu():
    print("1. Option 1")
    print("2. Option 2")
    print("3. exit")

display_menu()
display_welcome("Sam")
display_intro("Adriana Ferraro")
```