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 with different parameters and return value
- understand that a function can call another function
- understand the scope of variable
A Python function has the following syntax:

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

- **Function name**
- **Function arguments** (comma separated)
- **Colon**
- **Indentation** (either 1 tab or 4 spaces)
- **Return value**
- **Statements in the body of the function**
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
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
def display_intro():
    message = "Game of Nim"
    print(message)

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

display_intro()
print()
display_menu()
```

Game of Nim
1. Option 1
2. Option 2
3. exit
Functions with default parameters

- Functions may have default value for its parameters inside the round brackets.

```python
1 def greet(name, message = "Good morning!"):  
2     print("Hello",name + ', ' + message)
3    
4 greet("Grace")
5 greet("James","How do you do?")
```

- The parameter ‘name’ does not have a default value and is required (mandatory) during a call.

- The parameter ‘message’ has a default value of "Good morning!". So, it is optional during a call. If a value is provided, it will overwrite the default value.
Functions with default parameters

- Any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.

- This means that non-default arguments cannot follow default arguments. For example, if we had defined the function header above as:

```python
def greet(message = "Good morning!", name):
    print("Hello",name + ', ' + message)

greet("Grace")
greet("James","How do you do?")
```

SyntaxError: non-default argument follows default argument
Python - indentation

- Python programs are structured through indentation
  - Most programming languages use the concept of blocks of code. It is desirable that blocks of code are indented (a style requirement, not a language requirement).
  - 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.
  - If a block has to be more deeply nested, it is simply indented further to the right.
  - The statement that marks the beginning of an indented block ends with a colon.

```python
import blah
n = blahblahblah
n = n + blahblahblah
blahblahblahblahblahblahblah:  
    blahblahblah:
        c1 = blahblahblah
        c2 = blahblahblah
    blahblahblahblahblahblahblah:
        blahblahblah
        blahblahblah
print("The end")
```
Python code is structured through indentation (the skeleton of a program below contains no functions):
Each statement marking the beginning of an indented block ends with a colon (example program diagram contains two functions):

- Two function definitions at the top of the program

- Maybe some import statements here

```
def ....
    Function body

def ....
    Function body
```

```
Block 1

    Block 2

    Block 3

    Block 2 continued

    Block 1 continued
```
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)
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
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()
```

Game of Nim
by Adriana Ferraro

********************

Game of Nim
by Adriana Ferraro

********************
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
```
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
****************************
bye bye!
```
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**

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

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

message = get_winner_message("Sam")
display_winner_details(66, message)
message = get_winner_message("Helen")
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) - \n                      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.
In a Python program:

- functions can have different parameters or default values
- function can have a return value or not (the ‘None’)
- a function can make a call to another function
- the scope of variable needs to be understood
Examples of Python features used in this lecture

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

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")
```