COMPSCE 101
Principles of Programming

Lecture 7 – Defining functions
At the end of this lecture, students should be able to:

- define a function which accepts parameters and returns values
- make calls to functions which have been defined
- use excellent function names and variable names to ensure that the purpose of the function is clear
Recap

- From lecture 6
  - get user input from the keyboard
  - generate a random number
  - convert between types

```python
import random
dice1 = random.randrange(1, 7)
age = random.randrange(66, 99)

user_input = input("Enter age: ")
age = int(user_input)

cost = input("Enter cost $")
cost = float(cost)

price = 32.45
message = "Final price $" + str(price)
```
Functions are like small programs which perform useful tasks. So far we have used several Python built-in functions, e.g., `len()`, `min()`, `round()`, `max()`, `input()`.

On line 1, the program **makes a call** to the `min()` function, on line 2 the program **makes a call** to the `max()` function and on line 3 the program **makes a call** to the `len()` function.

All three functions **return** an integer (the result of the function code being executed). On lines 1 and 2, the returned value is printed. On line 3 the returned value is assigned to the variable, `length`. 
One of the aims when writing programs is to reuse code as much as possible.

Whenever we make a call to a function, the code inside the function definition is executed and the call we make is replaced by the result of the function (the value returned by the function).
Another aim when writing programs is to generalise the solution so it can be used for all similar problems.

```python
area = 5 * 10
print("Area of a rectangle with width 5 and height 10:", area)
```

Area of a rectangle with width 5 and height 10: 50

The above solution is not useful if we want to calculate the area of rectangles of different sizes. A more general (and more useful) solution:

```python
width = 5
height = 10
area = width * height
output_str = "Area of a rectangle with width " + str(width) + " and height " + str(height) + ":"
print(output_str, area)
```

Area of a rectangle with width 5 and height 10: 50
What is undesirable about the following code?

```python
import random

current_score = 0

dice1 = random.randrange(1, 7)
dice2 = random.randrange(1, 7)

current_score = current_score + dice1 + dice2

print("1.", "You threw a", dice1, "and a", dice2, " Score:", current_score)

dice1 = random.randrange(1, 7)
dice2 = random.randrange(1, 7)

current_score = current_score + dice1 + dice2

print("2.", "You threw a", dice1, "and a", dice2, " Score:", current_score)

#Continued on the next slide
```

1. You threw a 4 and a 6  Score: 10
2. You threw a 3 and a 5  Score: 18
3. You threw a 3 and a 2  Score: 23
4. You threw a 6 and a 6  Final Score: 35/48
#Continued from the previous slide

current_score = current_score + dice1 + dice2
print("3.", "You threw a", dice1, "and a", dice2, \
       "  Score:", current_score)

dice1 = random.randrange(1, 7)
dice2 = random.randrange(1, 7)
current_score = current_score + dice1 + dice2
print("4.", " You threw a ", dice1, " and a ", dice2, \
       "   Final Score: ", current_score, "/", 48, sep = ")

1. You threw a 4 and a 6  Score: 10
2. You threw a 3 and a 5  Score: 18
3. You threw a 3 and a 2  Score: 23
4. You threw a 6 and a 6  Final Score: 35/48
A Python function has the following syntax:

```python
def function_name(comma_separated_parameters):
    statements in the function
    return value_to_be_returned
```
Functions - example

The following function calculates the total number of minutes. The function is passed two parameters: the hours and the minutes.

```python
def get_minutes(hours, minutes):
    total = hours * 60 + minutes
    return total
```

The code in a function is not executed until the function is called:

```python
def get_minutes(hours, minutes):
    total = hours * 60 + minutes
    return total

total_minutes = get_minutes(3, 44)
print("1.", total_minutes, " minutes")
print("2.", get_minutes(5, 0), " minutes")
print("3.", get_minutes(11, 540), " minutes")
```

Three calls to the get_minutes() function (on lines 4, 6 and 7).
Functions – things to note

In the function call (line 4), there must be the **same number of arguments** passed to the function as the function requires (see the expected parameters on line 1 of the code). The order of the arguments is important.

In the program, the function definition (lines 1, 2 and 3) must occur before any of the calls to the function (line 4).

In the function definition (lines 1, 2 and 3), the **return statement** is the last statement (line 3) of the function.

```python
def get_minutes(hours, minutes):
    total = hours * 60 + minutes
    return total

total_minutes = get_minutes(3, 44)
print(total_minutes, " minutes")
```

224 minutes

This requirement will change – see lecture 9.
Functions – the return statement

|   | def get_minutes(hours, minutes):
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>total = hours * 60 + minutes</td>
</tr>
<tr>
<td>2</td>
<td>return total</td>
</tr>
<tr>
<td>3</td>
<td>total_minutes = get_minutes(3, 44)</td>
</tr>
<tr>
<td>4</td>
<td>print(total_minutes, &quot; minutes&quot;)</td>
</tr>
<tr>
<td>5</td>
<td>224 minutes</td>
</tr>
</tbody>
</table>

- In the function definition (lines 1, 2 and 3), the **return statement** is always the last statement (line 3). When the return statement is reached, the function stops executing returning the value (total in the example above) to the function call. Control goes back to the function call (the right hand side of line 4) and the program continues executing at line 4 followed by line 5.

- All the statements inside the function (in the **body of the function**) are **indented** (either one tab or 4 spaces). This is the body of the function.
The following function (lines 1, 2, 3) converts degrees Celsius to degrees Fahrenheit using the formula:

\[ ^\circ F = (^\circ C \times 1.8) + 32 \]

```python
def celsius_to_f(celsius):
    fahrenheit = celsius * 9 / 5 + 32
    return fahrenheit

celsius = 34
print(1, "celsius", celsius, "= fahrenheit", celsius_to_f(celsius))

celsius = 15
print(2, "celsius", celsius, "= fahrenheit", celsius_to_f(celsius))

celsius = 21
print(3, "celsius", celsius, "= fahrenheit", celsius_to_f(celsius))
```

1 celsius 34 = fahrenheit 93.2
2 celsius 15 = fahrenheit 59.0
3 celsius 21 = fahrenheit 69.8
Functions – use clear function names

When defining functions always use self-documenting function names and, as in all code, use self-documenting variable names. You should always write code which is easy to read and understand.

All functions should be clear and aim to perform only one task.
Exercise

- Define the function1() function which is passed three whole numbers. The function returns the total of the two bigger numbers.

```python
print("1.", function1(1, 2, 3))
print("2.", function1(11, 12, 3))
print("3.", function1(6, 2, 5))
```

1. 5
2. 23
3. 11
Exercise

- Define the function2() function which is passed two strings. The function returns the length of the shorter of the two strings.

```python
print("1.", function2("Flibbertigibbet", "Rigmarole"))
print("2.", function2("Mollycoddle", "Cat"))
print("3.", function2("Skullduggery", "Canoodle"))
```

1. 9
2. 3
3. 8
Exercise

- Define the function3() function which is passed one string. The function returns a string made up of the last character followed by the first character (both uppercase characters).

```python
print("1.", function3("Crudivore"))
print("2.", function3("Ornery"))
print("3.", function3("Brouhaha"))
```

1. EC
2. YO
3. AB
Define the `required_boxes()` function which is passed a total number of items and the maximum number of items which fit into one box. The function returns the total number of boxes required (any leftovers always require an extra box).

```python
boxes_needed1 = required_boxes(30, 16)
bboxes_needed2 = required_boxes(30, 3)
bboxes_needed3 = required_boxes(30, 10)

print("1.", "Boxes:", boxes_needed1)
print("2.", "Boxes:", boxes_needed2)
print("3.", "Boxes:", boxes_needed3)
```

1. Boxes: 2
2. Boxes: 10
3. Boxes: 3
Exercise

Complete the two functions in the following program. The following program gets a first name from the user (prompt is "Enter name: ") and then removes a random letter from the name. The resulting name is printed.

```python
def get_first_name():

def remove_random_letter(name):

first_name = get_first_name()
version1 = remove_random_letter(first_name)
version2 = remove_random_letter(first_name)
version3 = remove_random_letter(first_name)

print("1.", version1)
print("2.", version2)
print("3.", version3)
```

Enter name: Adriana
1. Adriaa
2. Adrina
3. Ariana
In a Python program:

- functions which accept parameters and return values can be defined
- calls to functions which have been defined are made (below the function definition)
- we must use meaningful names and variable names to ensure that the purpose of the function is clear
- Each function performs one task
Examples of Python features used in this lecture

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

def celsius_to_f(celsius):
    farenheit = celsius * 9 / 5 + 32
    return farenheit

dice_throw = get_dice_total()
farenheit = celsius_to_f(34)
```
print(1, (1, 2, 3) )
print(2, (11, 12, 3) )
print(3, (6, 2, 5) )
print(1, ("Flibbertigibbet", "Rigmarole"))
print(2, ("Mollycoddle", "Malarkey"))
print(3, ("Skullduggery", "Canoodle"))
print(1,  
    ("Crudivore")  )
print(2,  
    ("Ornery")  )
print(3,  
    ("Brouhaha")  )