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
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: ")
user_age = int(user_input)

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

price = cost + 32.45
message = "Final price "+ str(price)
```
Python built-in functions

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 (i.e., replaced by the value returned by the function).

```
1 name = input("Enter name: ")
2 age = int(input("Enter age: "))
3 bday_month = input("Enter birthday month: ")
```
Generalise

Another aim when writing programs is to generalise the solution so it can be used over and over with different values.

```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
Exercise

What is undesirable about this code (continues onto the next slide)?

```python
import random

current_score = 0
num = 1
dice1 = random.randrange(1, 7)
dice2 = random.randrange(1, 7)
current_score = current_score + dice1 + dice2
result_str = str(num) + ". You threw a " + str(dice1) + " and a " + str(dice2) + " Score: " + str(current_score)
print(result_str)

num = 2
dice1 = random.randrange(1, 7)
dice2 = random.randrange(1, 7)
current_score = current_score + dice1 + dice2
result_str = str(num) + ". You threw a " + str(dice1) + " and a " + str(dice2) + " Score: " + str(current_score)
print(result_str)
```

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  Score: 35
Exercise continued

```python
num = 3
dice1 = random.randrange(1, 7)
dice2 = random.randrange(1, 7)
current_score = current_score + dice1 + dice2
result_str = str(num) + ". You threw a " + str(dice1) + ", and a " + str(dice2) + " Score: " + str(current_score)
print(result_str)

num = 4
dice1 = random.randrange(1, 7)
dice2 = random.randrange(1, 7)
current_score = current_score + dice1 + dice2
result_str = str(num) + ". You threw a " + str(dice1) + ", and a " + str(dice2) + " Score: " + str(current_score)
print(result_str)
```

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  Score: 35
A Python function has the following syntax:

```python
def function_name(comma_separated_parameters):
    statements in the function
    return value_to_be Returned
```

- **'def'**
- Function name
- Function parameters
- colon
- Indentation (either 1 tab or 4 spaces)
- 'return'
- Return value
- Statements in the body of the function.
Functions - example

The function defined below 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")
```

There are 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 (the expected parameters are on line 1 inside the parentheses). 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

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 (the variable, 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 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
# Celsius to Fahrenheit function

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

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

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

- 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 clear and easy to understand.

- All functions should be clear and aim to perform only one task.
Define the get_result1() function which is passed three whole numbers. The function returns the sum of the two bigger numbers.

```python
print("1.", get_result1(1, 2, 3))
print("2.", get_result1(11, 12, 3))
print("3.", get_result1(6, 2, 5))
```
Exercise

Define the get_result2() function which is passed two strings. The function returns the number of characters in the longer of the two strings.

```python
print("1.", get_result2("Flibbertigibbet", "Rigmarole"))
print("2.", get_result2("Mollycoddle", "Cat"))
print("3.", get_result2("Skullduggery", "Canoodle"))
```

1. 15
2. 11
3. 12
Exercise

Define the get_result3() function which is **passed** one string. The function **returns** a string made up of the last character followed by the first character (both in uppercase characters).

```python
print("1.", get_result3("crudivorE"))
print("2.", get_result3("OrnerY"))
print("3.", get_result3("brouhaha"))
```

1. EC
2. YO
3. AB
Exercise

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).

boxes_needed1 = required_boxes(30, 16)
boxes_needed2 = required_boxes(30, 3)
boxes_needed3 = required_boxes(30, 10)

print("1.", "Boxes: ", boxes_needed1)
print("2.", "Boxes: ", boxes_needed2)
print("3.", "Boxes: ", boxes_needed3)
In a Python program:

- functions which accept parameters and return values can be defined
- calls to functions which have been defined cause the code inside the function to be executed
- 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

def get_dice_total():
    dice1 = random. randrange(1, 7)
    dice2 = random. randrange(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)