Learning outcomes

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

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().

  ```
  print(min(5, 78, 15))  # 5
  print(max(5, 78))      # 78
  length = len("ABCDEFG") # 7
  print(length)
  ```

- 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.

Reuse code

- One of the aims when writing programs is to reuse code as much as possible.

  ```
  name = input("Enter name: ")
  age = int(input("Enter age: "))
  bday_month = input("Enter birthday month: ")
  ```

- 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.

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:

Not a good way to program!

The area is calculated as:

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

Example

What is undesirable about the following code?

```
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)
dice1 = random.randrange(1, 7)
dice2 = random.randrange(1, 7)
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, 
"  Score:", current_score)
```

The output is as follows:

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

Simplified Example

```
import random
def throw_2_dices(num, current_score):
dice1 = random.randrange(1, 7)
dice2 = random.randrange(1, 7)
current_score = current_score + dice1 + dice2
print(num, "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(num, "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(num, "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(num, "You threw a", dice1, "and a", dice2, 
"  Score:", current_score)
```

The output is as follows:

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
Syntax of a Python function

* 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
```

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

```python
total_minutes = get_minutes(5, 0)
print("2.", total_minutes, " minutes")
```

```python
total_minutes = get_minutes(11, 540)
print("3.", total_minutes, " minutes")
```

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.

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 (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:

\[ F = \left(\frac{C \times 9}{5}\right) + 32 \]

```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))

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.

Examples of Python features used in this lecture:

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

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

dice_throw = get_dice_total()
fahrenheit = celsius_to_f(34)
```