Class exercises: User-defined functions

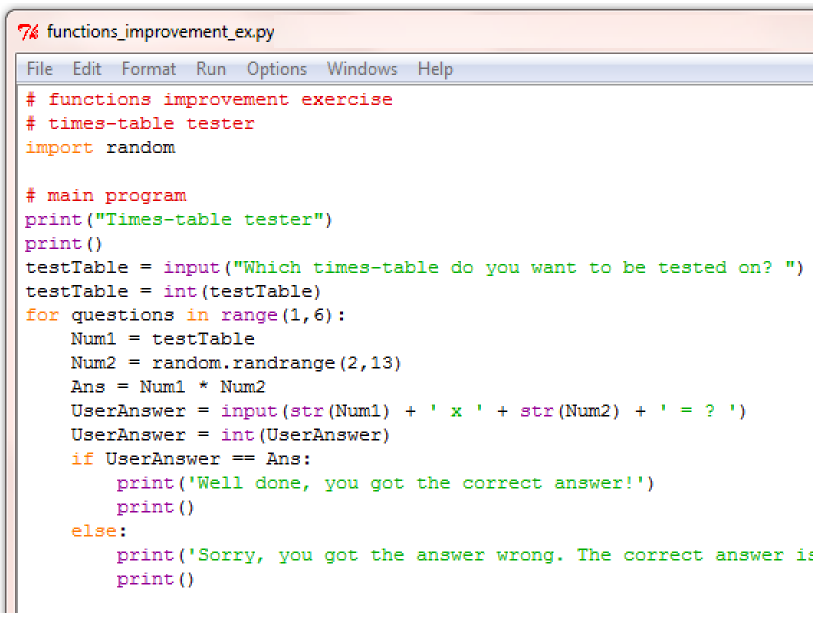
These tasks are designed to help you practice the skills and knowledge you have developed. There are exercises to help you revise and develop your understanding and also to challenge you further.

A function is a block of reusable code that is used to perform some specific action. Functions provide a way of breaking up more complex programs into more manageable blocks and also make it easier to re-use code.

As you have seen, Python provides many built-in functions but you can also create your own. These functions are called *user-defined functions*.

Improving a piece of code

Task One:
Open the Python file called functions_improvement_ex.py and refactor it so that it uses a function to present the questions and display whether the answer is correct or not.


functions_improvement_ex.py
File Edit Format Run Options Windows Help

```
# functions improvement exercise
# times-table tester
import random

# main program
print("Times-table tester")
print()
testTable = input("Which times-table do you want to be tested on? ")
testTable = int(testTable)
for questions in range(1,6):
    Num1 = testTable
    Num2 = random.randrange(2,13)
    Ans = Num1 * Num2
    UserAnswer = input(str(Num1) + ' x ' + str(Num2) + ' = ? ')
    UserAnswer = int(UserAnswer)
    if UserAnswer == Ans:
        print('Well done, you got the correct answer!')
        print()
    else:
        print('Sorry, you got the answer wrong. The correct answer is
        print()
```

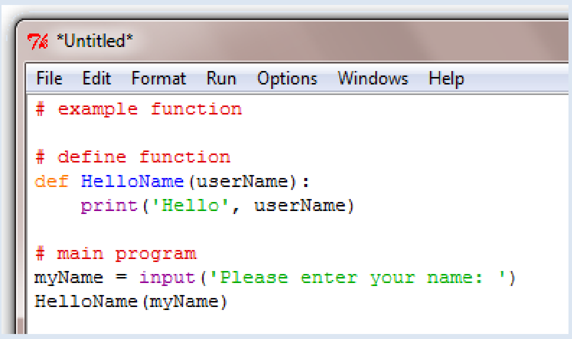
Defining a function

Function blocks begin with the keyword `def` followed by the function name, parentheses ( ), and finally a colon.

```
def functionName():
```

Any parameters that the function needs are named in the parentheses.

```
def functionName(parameterName):
```

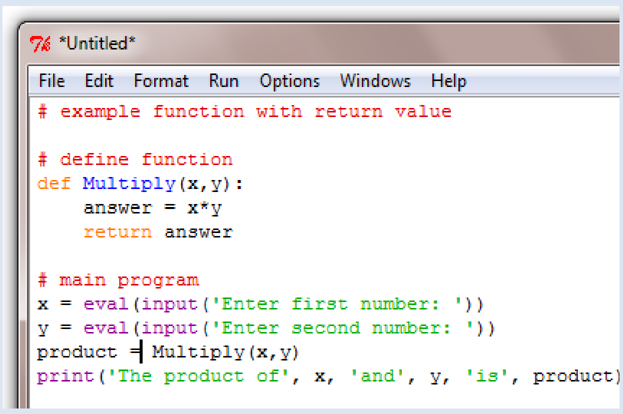


```
# example function

# define function
def HelloName(userName):
    print('Hello', userName)

# main program
myName = input('Please enter your name: ')
HelloName(myName)
```

The return value

Another thing that a function might do is work out an answer based on parameters passed to it and then **return** the answer to use.

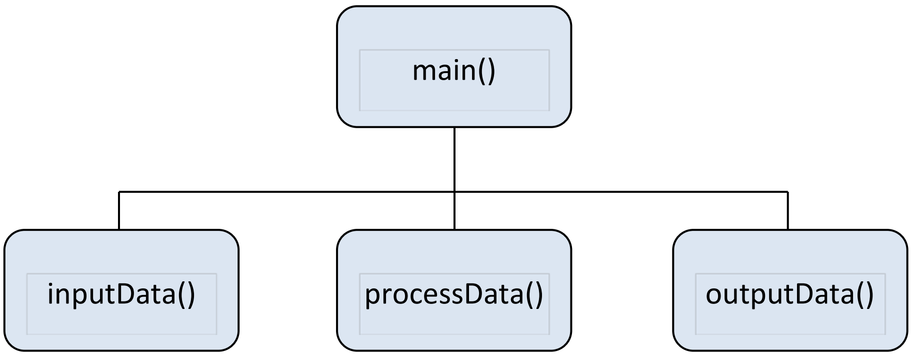
To do this, use the **return** keyword:

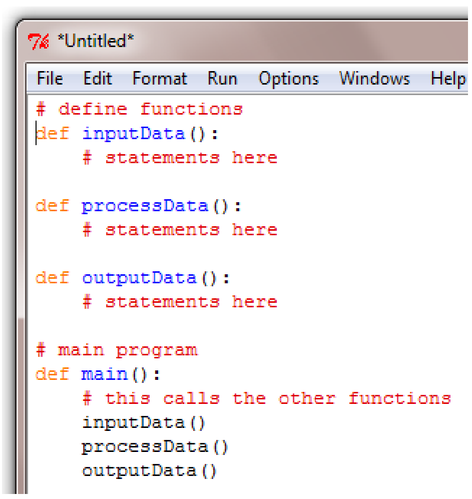


```
# example function with return value

# define function
def Multiply(x,y):
    answer = x*y
    return answer

# main program
x = eval(input('Enter first number: '))
y = eval(input('Enter second number: '))
product = Multiply(x,y)
print('The product of', x, 'and', y, 'is', product)
```

Passing data between functions

Many programs require getting input from a user, processing it in some way, and then outputting the results. This can be shown in a simple hierarchy chart as follows:





```
# define functions
def inputData():
    # statements here

def processData():
    # statements here

def outputData():
    # statements here

# main program
def main():
    # this calls the other functions
    inputData()
    processData()
    outputData()
```

Group programming

Task Two:

An online bank wants a program to collect information from its customers when they register on its website. When a user firsts registers, the program should collect their first name, last name, title (Mr, Mrs, Ms etc), house number/name and street name, town and post code. These details are then passed on so that they can be processed and the person's title, first name and last name only are returned, so that it can display a suitable personalised message on screen (e.g. Welcome Mrs. ...) telling them that registration has been successful.

(a) In a group of three, agree a hierarchy chart for the three elements to this program and what each element should do.
(b) Divide the three elements between you so that each person writes the function for one element and tests it.
(c) Put the three functions together into one program (without making any changes to the functions at this point) and write a short main program to call each of them in turn. Test your overall program with suitable test data. If the program does not work correctly, then the person responsible for the relevant function should adapt their code.

Questions to consider from the group programming task:

1 What must be 'agreed' between the functions in order for the complete program to work correctly?
2 How much does one function need to know about the other functions' code in order for it to be able to work correctly?

Class exercises

- Create pseudocode or a flowchart or a structure chart first - plan your solution on paper before attempting it.
- Create a set of test data - select values you will enter to test the program and know beforehand what you expect the answers to be.
- Write the program - write the program in Python using the pseudocode to assist you and the test data to ensure the program functions correctly.

Revision exercises

Attempt these tasks if you need to build your confidence and understanding of using functions.

1. Write and test a function OutputSymbols, which takes two parameters: an integer *n* and a character symbol. The function is to display, on the same line, the symbol *n* times. For example, the call OutputSymbols(5, '#') should display #####.
2. Using functions, write a program that asks the user to enter an odd number, validates that the number is odd and then prints an inverted pyramid of stars based on that number. For example, entering the value 5 will produce:

```
* * * * *
  * * *
    *
```

3. Write and test a function Sort, which takes two integers as parameters and returns them in ascending order (i.e. if x > y, then the function should return y, x).
4. Write a program to convert temperatures from Fahrenheit to Celsius. The function should take one parameter (the temperature in Fahrenheit) and return a result (the temperature in Celsius).
   The formula for the conversions is:
   Celsius = (Fahrenheit - 32) * (5/9)

Development exercises

Attempt these tasks if you are confident in your understanding of functions but feel you need more practice.

5. Write a function that returns the total number of seconds, calculated from a whole number of hours, minutes and seconds provided as 3 parameters.
6. Using functions, write a program that asks the user to enter an odd number, validates that the number is odd, and then displays a diamond of stars based on that number. For example, entering the value 5 will produce:

```
  *
 ***
*****
 ***
  *
```

7. Write a currency converter program that uses functions. It should ask the user which currency to convert to and from and how much they want to convert. It should convert between the Euro, US dollar and British pound. Use these conversion rates:

£1 = $1.601	£1 = €1.229	€1 = £0.814	€1 = $1.302	$1 = £0.625	$1 = €0.768

Stretch and challenge exercises

Attempt these tasks if you feel you have mastered functions and want to tackle tougher problems.

8. Write a program to convert measurements between feet and inches and metres and centimetres. The program should ask the user whether they wish to convert from metres and centimetres to feet and inches, or from feet and inches to metres and centimetres. Create two functions (one for conversion one way, and the other for conversion the other way) and each should take two parameters (the number of feet and inches (one parameter for each) or the number of metres and centimetres (one parameter for each).
   The conversions are:

1 inch = 2.54 cm	1 foot = 30.48 cm
1 cm = 0.397 inches	1 metre = 39.37 inches

9. The game ‘Last one loses’ is played by two players and uses a pile of *n* counters. Players take turns at removing 1, 2 or 3 counters from the pile. The game continues until there are no counters left and the winner is the one who does not take the last counter.
   Using functions, write a program to allow the user to specify *n* in the range 10 - 50 inclusive. The computer acts as one player, playing at random.
10. Develop your program for 'Last one loses' so that the computer plays at random until 5 or fewer counters remain. The computer should then play to win.