Code Clock

Tutorial 4: Functions

Learn.to.code

Programming with Python

@ QUB

Functions

A function is a block of organised, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

As you already know, Python gives you many built-in functions like `print(),` etc. but you can also create your own functions. These functions are called *user-defined functions.*

Defining a Function

A function is a reusable block of programming statements designed to perform a certain task. To define a function, Python provides the **`def`** keyword.

Enter the following code to create a simple function (def) entitled codeClock and then run the function by calling it the below the function

```
def codeClock():
  print("Welcome to Code Clock Week 4")

#This will ensure the function is 'called' i.e runs the function in the program

codeClock()
```

Function Parameters

It is possible to define a function to receive one or more parameters (also called arguments) and use them for processing inside the function block.

Parameters/arguments may be given suitable formal names. The **`printMyName`**`()` function below is now defined to receive a string parameter called name. Inside the function, the **`print`**`()` statement is modified to display a person's name addressed to the received parameter.

Enter the following code to create a simple def entitled printMyName which takes a parameter name and concatenates it with a string within a print function

```
def printMyName(name):
  print("Welcome to Code Clock Week 4" + name)

printMyName("Errol")
```

The function parameters can have additional annotation to specify the type of the parameter using parameter: type syntax. For example, the following annotates the parameter type string.

Enter the following code to create a def which specifies the parameter type accepted by the function

```
def printMyName(name: str):
  print(“Welcome to Code Clock Week 4” + name)
printMyName(“Errol”)
printMyName(123)#This should throw an error
```

Multiple Parameters

A function can have multiple parameters of the same data types.

Enter the following code which takes three parameters (names) and prints them out through a print function

```
def printNames(name1, name2, name3):
  print(“Welcome to Code Clock Week 4\n ” + name1 +”\n” + name2 +
“\n” + name3)

printNames (“David”,“Michael”,“Bradley”)
```

…and of different types.

Enter the following code which takes three parameters of varied types and prints them out through a print function

```
def printTrainerDetails(name, age, gender):
  print(“Your trainer today is\n- Name: ” + name + “\n-age” + age +
“\n-gender” + gender)

printTrainerDetails (“Errol Martin”,“45”,“Male”)
```

Unknown Number of Parameters

A function can have specify to receive a unknown numbers of parameters. This is achieved by placing an * in front of the parameter in brackets

Enter the following code which takes an unknown number of parameters (names) and prints them out through a print function

```
def printName(*names):
    print(“Welcome to Code Clock Week 4\n ” + name[0] + “\n” +
    name[1] + “\n” + name[2])

printNames (“Rosie”,”Sam”,”Autumn”)
```

Default Parameters

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

Enter the following code which takes an four parameters of different types, one of which has a default value

```
def printTrainerDetails(name, age, gender, course= “Course:
  Programming with Python”):
  print(“Name:” + name)
  print(“Age:” + age)
  print(“Gender” + gender)
  print(course)
printTrainerDetails (“Errol Martin”,“45”,“Male”)

#What happens when you passed the fourth parameter
printTrainerDetails (“Errol Martin”,“45”,“Male”,“Programming with
C#”))
```

Return Statement (single)

The statement return enables a functions to send back a value(s) to the function call.

Enter the following code which creates a function taking two integer parameters and returns the total the function call

```
def simpleCalc(num1, num2)
  total = num1+num2
  return total
print(“The sum of 12 and 14 is ” + str(simpleCalc(12,14)))
```

Enter the following code which create a function to convert an user defined parameter inches into cm and returns two values through the return statement

```
def convertInchToCm(inch):
   cm = inch * 2.54
   return cm, "cm"

inch = int(input("Enter inches: "))
value, unit = convertInchToCm(inch)
print("Conversion =" + value + unit)
```

Global vs. Local variables

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope.

Enter the following code which highlights the difference between local and global variables within the context of a function

```
total = 0; # This is global variable.
def sum(num1,num2):
   #Add both the parameters and return them.
   total = arg1 + arg2; #Here total is local variable.
   print(“Inside the function local total :” + total)
   return total;

#Now you can call sum function
sum(10, 20);
print(“Outside the function global total: ” + total)
```

Challenge:

1. Write a function, and call it, which prints a parameter containing today's date i.e. 25th Feb 2023
2. Write a function, and call it, which when called prints the number 1 to 10
3. Write a function, and call it, which calculates and prints the average of two numbers passed as parameters.
4.
    a) Write a function, and call it, which takes three integer parameters and calculates the volume of a rectangle, returning both the volume and unit of measurement i.e. "cm^3" and prints in a suitable output statement.
    b) Adapt the above to enable the user to enter the three dimensions.
5.
    a) Write a function which calculates and returns both the simple interest AND amount to repay to replay on a user-defined loan amount and user-defined number of years with an interest rate of 4.5%. The code should ensure that the user enters a loan amount of no more than £30,000 and no less than £1000. The minimum period should also be no less than 2 years. Both of these values should be printed within the function call statement.
    b) Create another function which now calculates compound interest for both user-defined loan amount and number of years.
    c) Adapt the code to enable the user to choose whether to calculate either simple interest or compound interest
    d) Adapt the code to enable the user to calculate either simple or compound interest more than once.
    e) Create a menu which enables the user to choose either:
       1. Calculate Simple Interest
       2. Calculate Compound Interest
       3. Stats*
       This menu will enable the user to choose which function to call.
       *This should print out the number of times the interest functions have been called during that a particular session.
    f) Adapt the code to create another function which handles after the input of these values

6. Create a registration function which enables an user to register within an application using their
    - first name,
    - surname,
    - date of birth and
    - gender.

   This registration function must also:

- validate the password to ensure it meets the following criteria.
  - must not contain the word password
  - must be between 9 and 15 characters in length
  - must contain a Upper case letter
  - must contain a Lower case letter
  - must contain one of the following special characters:
  - @
  - #
  - £
  - &
  - ?

  The registration function should also call a function which generates an username which uses the first initial of the registrants first name, their surname and randomly generate 3-digit number to create a username in the form "emartin680". The user should receive confirmation of both their newly generated username and also confirmation that the password was successfully.