4 Functions

4.1 Introduction

As the words suggests, *functions* are a piece of code that have a specific function or purpose. As an analogy, if a human is a computer program, then the mind can be considered to be a function, which has purpose of thinking, eyes can be another function, which have a purpose of seeing. These functions are called upon by the human when needed.

Similarly, in case of a computer program, functions are a piece of code, that perform a specific task, when called upon by the program. Instead of being defined as a function, the piece of code can also be used directly whenever it is needed in a program. However, defining a frequently-used piece of code as a function has the following benefits:

1. It reduces the number of lines of code, as the lines of code need to be written just once in the function definition. Thereafter, the function is called by its name, wherever needed in the program. This makes the code compact, and enhances readability.
2. It makes the process of writing code easier, as the user needs to just type the name of the function, wherever it is needed, instead of pasting lines of code.
3. It can be used in different programs, thereby saving time in writing other programs.

To put it more formally, a function is a piece of code that takes arguments (if any) as input, performs computations or tasks, and then returns a result or results.

4.2 Defining a function

Look at the function defined below. It asks the user to input a number, and prints whether the number is odd or even.

```
#This is an example of a function definition

#A function definition begins with the 'def' keyword followed by
#Note that 'odd_even()' is the name of the function below.
def odd_even():
    num = int(input("Enter an integer:"))
```

```
    if num%2==0:
        print("Even")
    else:
        print("Odd")   #Function definition ends here

print("This line is not a part of the function as it is not inde
```

```
This line is not a part of the function as it is not indented
```

Note that the function is defined using the `def` keyword. All the lines within the function definition are indented. The indentation shows the lines of code that below to the function. When the indentation stops, the function definition is considered to have ended.

Whenever the user wishes to input a number and print whether it is odd or even, they can call the function defined above by its name as follows:

```
odd_even()
```

```
Enter an integer:5
Odd
```

In Python, empty parentheses are used when defining a function, even if it doesn't take any parameters. This is a syntactic requirement to differentiate between variables and functions. It helps Python understand that you are defining a function, not just referencing a variable.

4.3 Parameters and arguments of a function

Note that the function defined above needs no input when called. However, sometimes we may wish to define a function that takes input(s), and performs computations on the inputs to produce an output. These input(s) are called parameter(s) of a function. When a function is called, the value(s) of these parameter(s) must be specified as argument(s) to the function.

4.3.1 Function with a parameter

Let us change the previous example to write a function that takes an integer as an input argument, and prints whether it is odd or even:

```
#This is an example of a function definition that has an argume
def odd_even(num):
    if num%2==0:
```

```
        print("Even")
    else:
        print("Odd")
```

We can use the function whenever we wish to find a number is odd or even. For example, if we wish to find that a number input by the user is odd or even, we can call the function with the user input as its argument.

```
number = int(input("Enter an integer:"))
odd_even(number)
```

```
Enter an integer:6
Even
```

Note that the above function needs an argument as per the function definition. It will produce an error if called without an argument:

```
odd_even()
```

```
---------------------------------------------------------------------
TypeError                                 Traceback (most recent
<ipython-input-8-d86a5f720e3b> in <module>
----> 1 odd_even()

TypeError: odd_even() missing 1 required positional argument: 'nu
```

4.3.2 Function with a parameter having a default value

To avoid errors as above, sometimes is a good idea to assign a default value to the parameter in the function definition:

```
#This is an example of a function definition that has an argume
def odd_even(num=0):
    if num%2==0:
        print("Even")
    else:
        print("Odd")
```

Now, we can call the function without an argument. The function will use the default value of the parameter specified in the function definition.

```
odd_even()
```

```
Even
```

4.3.3 Function with multiple parameters

A function can have as many parameters as needed. Multiple parameters/arguments are separated by commas. For example, below is a function that inputs two strings, concatenates them with a space in between, and prints the output:

```
def concat_string(string1, string2):
    print(string1+' '+string2)
```

```
concat_string("Hi", "there")
```

```
Hi there
```

4.3.4 Practice exercise 1

Write a function that prints prime numbers between two real numbers - `a` and `b`, where `a` and `b` are the parameters of the function. Call the function and check the output with a = 60, b = 80.

Solution:

```
def prime_numbers (a,b=100):
    num_prime_nos = 0

    #Iterating over all numbers between a and b
    for i in range(a,b):
        num_divisors=0

        #Checking if the ith number has any factors
        for j in range(2, i):
            if i%j == 0:
                num_divisors=1;break;

        #If there are no factors, then printing and counting the
        if num_divisors==0:
            print(i)
prime_numbers(60,80)
```

```
61
67
```

```
71
73
79
```

4.4 Functions that return objects

Until now, we saw functions that print text. However, the functions did not `return` any object. For example, the function `odd_even` prints whether the number is odd or even. However, we did not save this information. In future, we may need to use the information that whether the number was odd or even. Thus, typically, we return an object from the function definition, which consists of the information we may need in the future.

The example `odd_even` can be updated to return the text “odd” or “even” as shown below:

```
#This is an example of a function definition that has an argume
def odd_even(num=0):
    if num%2==0:
        return("Even")
    else:
        return("Odd")
```

The function above returns a string “Odd” or “Even”, depending on whether the number is odd or even. This result can be stored in a variable, which can be used later.

```
response=odd_even(3)
response
```

```
'Odd'
```

The variable `response` now refers to the object where the string “Odd” or “Even” is stored. Thus, the result of the computation is stored, and the variable can be used later on in the program. Note that the control flow exits the function as soon as the first `return` statement is executed.

Figure 4.1 below shows the terminology associated with functions.

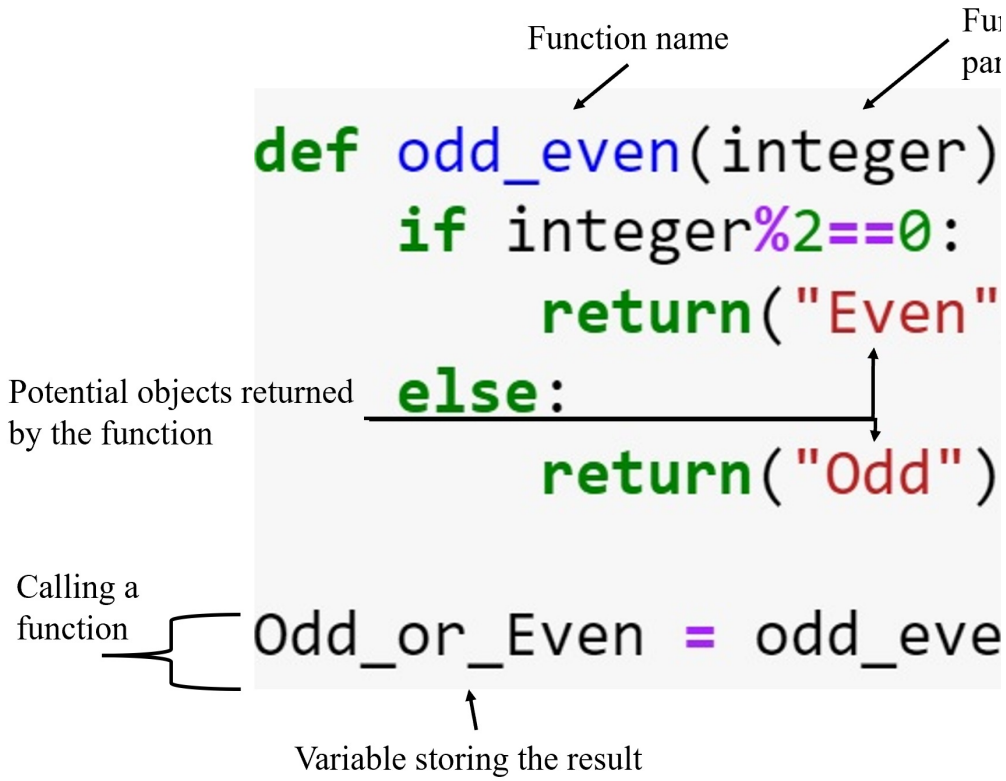

Figure 4.1: Terminology associated with functions

4.5 Global and local variables with respect to a function

A variable defined within a function is local to that function, while a variable defined outside the function is global with respect to that function. In case a variable with the same name is defined both outside and inside a function, it will refer to its global value outside the function and local value within the function.

The example below shows a variable with the name `var` referring to its local value when called within the function, and global value when called outside the function.

```
var = 5
def sample_function(var):
    print("Local value of 'var' within 'sample_function()'= ",va

sample_function(4)
print("Global value of 'var' outside 'sample_function()' = ",va
```

```
Local value of 'var' within 'sample_function()'=  4
Global value of 'var' outside 'sample_function()' =  5
```

4.6 Built-in python functions

So far we have seen user-defined functions in this chapter. These functions were defined by us, and are not stored permanently in the python compiler. However, there are some functions that come built-in with python and we can use them directly without defining them. These built-in functions can be see here. For example the built-in function `max()` computes the max of numeric values:

```
max(1,2,3)
```

3

Another example is the `round()` function that rounds up floating point numbers:

```
round(3.7)
```

4

4.7 Python libraries

Other than the built-in functions, python has hundreds of thousands of libraries that contain several useful functions. These libraries are contributed by people around the world as python is an open-source platform. Some of the libraries popular in data science, and their purposes are the following:

1. NumPy: Performing numerical operations and efficiently storing numerical data.
2. Pandas: Reading, cleaning and manipulating data.
3. Matplotlib, Seaborn: Visualizing data.
4. SciPy: Performing scientific computing such as solving differential equations, optimization, statistical tests, etc.
5. Scikit-learn: Data pre-processing and machine learning, with a focus on prediction.
6. Statsmodels: Developing statistical models with a focus on inference

A library can be imported using the `import` keyword. For example, a NumPy library can be imported as:

```
import numpy as np
```

Using the `as` keyboard, the NumPy library has been given the name `np`. All the functions and attributes of the library can be called using the *'np.'* prefix. For example, let us generate a sequence of whole numbers upto `10` using the NumPy function arange():

```
np.arange(8)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7])
```

Generating random numbers is very useful in python for performing simulations (we'll see in later chapters). The library random is used to generate random numbers such as integers, real numbers based on different probability distributions, etc.

Below is an example of using the `randint()` function of the library for generating random numbers in [a, b], where `a` and `b` are integers.

```
import random as rm
rm.randint(5,10) #This will generate a random number in [5,10]
```

```
7
```

4.7.1 Practice exercise 2

Generate a random number between [-5,5]. Do this 10,000 times. Find the mean of all the 10,000 random numbers generated.

Solution:

```
import random as rm
counter = 0
for i in range(10000):
    counter = counter + rm.uniform(-5,5)
print("Mean is:", counter/10000)
```

```
Mean is: 0.061433810226516616
```