A function is a block of organized, 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 Funcion

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword *def* followed by the function name and parentheses ().
- Any input parameters or arguments should be placed within these parentheses.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax

```
def functionName():
    statement1
    statement2
    return [expression]
```

Calling a Function

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.

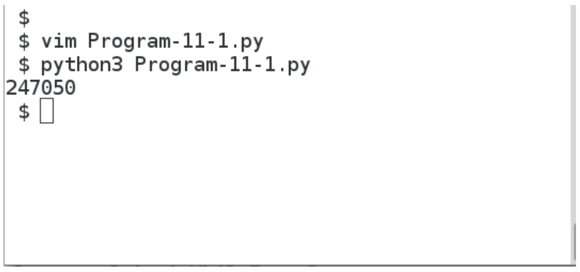
Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

```
>>> functionName()
```

Example 1 print the sum of all even numbers in a given range.

```
def isEven(num):
    return (num % 2 == 0)

def sumOfEvenNumbers(st_val, limit):
    sumEvenNums = 0
    for num in range(st_val, limit + 1):
        if (isEven(num)):
            sumEvenNums += num
    return sumEvenNums

print(sumOfEvenNumbers(100,999))
```

Example 2 print all palindrome numbers, which contains all even digits.

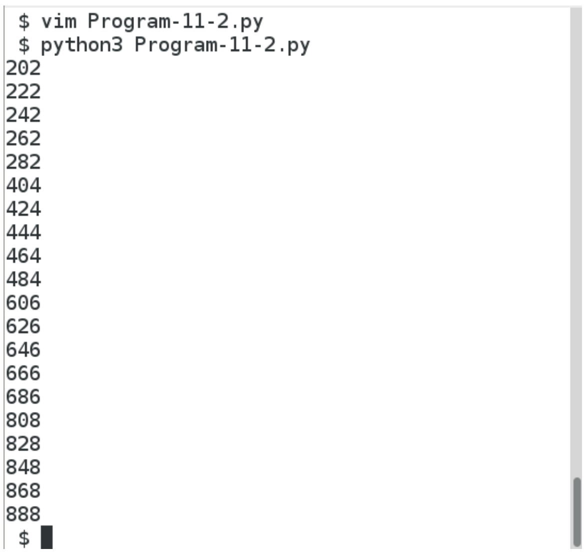


```
 $
 $ vim Program-11-1.py
 $ python3 Program-11-1.py
247050
 $ 
```

Figure 1: Output

```

# function to get the reverse of the given number

def reverse(num):
    rev_num = 0
    while (num > 0):
        rem = num % 10
        rev_num = (rev_num * 10) + rem
        num //= 10
    return rev_num

# function to check the number is palindrome
```

```
def isPalindrome(num):
    return (reverse(num) == num)


# function to check all digits are even
def allEvenDigits(num):
    while (num > 0):
        rem = num % 10
        if (rem % 2 != 0):
            return False
        num //= 10
    return True

def allPalindromes(n1, n2):
    for num in range(n1, n2 + 1):
        if (isPalindrome(num) and allEvenDigits(num)):
            print(num)

# function call
allPalindromes(100,999)
```



```
 $ vim Program-11-2.py
 $ python3 Program-11-2.py
202
222
242
262
282
404
424
444
464
484
606
626
646
666
686
808
828
848
868
888
 $
```

Figure 2: Output

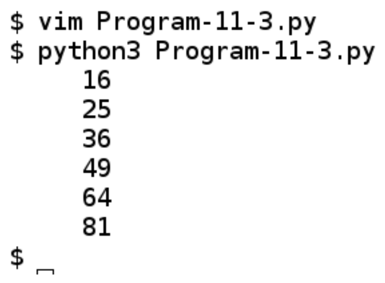
Returning Multiple Values

A function can return exactly one value, or we should better say one object. An object can be a value of any type like., *integer, float* or *boolean* and it can also be a *list* or a *tuple*. So, if we have to return more than one value, we can use *list* or *tuple* for returning multiple values.

Example:

A program to print all two digit perfect square numbers.

```python
def isPerfectSquare(n):
    '''This function returns
          a number and a boolen value'''
    f = 1
    while f * f < n:
        f += 1
    return f, f * f == n

def generatePerfectSquares(LO, HI):
    for num in range(LO, HI):
        val, status = isPerfectSquare(num)
        if status:
            print("{0:8}".format(num))


generatePerfectSquares(10, 100)
```



```
$ vim Program-11-3.py
$ python3 Program-11-3.py
     16
     25
     36
     49
     64
     81
$ _
```

Figure 3: Output