13 Functions

"Think modular, think of functions..."

Contents

HkgBooks

KanNotes

What are Functions?

- Python function is a block of code that performs a specific and well-defined task.
- Two main advantages of function are:
  - (a) They help us divide our program into multiple tasks. For each task we can define a function. This makes the code modular.
  - (b) Functions provide a reuse mechanism. The same function can be called any number of times.
- There are two types of Python functions:
  - (a) Built-in functions - Ex. **len( )**, **sorted( )**, **min( )**, **max( )**, etc.
  - (b) User-defined functions
- Given below is an example of user-defined function. Note that the body of the function must be indented suitably.

```
# function definition
def fun( ) :
    print('My opinions may have changed')
    print('But not the fact that I am right')
```

- A function can be called any number of times.

```
fun( )          # first call
fun( )          # second call
```

- When a function is called, control is transferred to the function, its statements are executed and control is returned to place from where the call originated.
- Python convention for function names:
  - Always use lowercase characters
  - Connect multiple words using _

    Example: cal_si( ), split_data( ), etc.
- A function can be redefined. While calling the function its latest definition will be called.

HkgBooks

- Function definitions can be nested. When we do so, the inner function is able to access the variables of outer function. The outer function has to be called for the inner function to execute.

```
def fun1( ) :
    print('Reached fun1')
    def fun2( ) :              # nested definition
        print('Inner avatar')
    print('Outer avatar')
    fun2( )

fun1( )                        # ok
fun2( )                        # cannot call inner function from here
print(type(fun1))              # nested call
```

- Suppose we wish to develop a function **myrandom( )** to generate random numbers. While executing this function we wish to check whether a number is a prime number or not. We can do so by defining a function **isprime( )**. But we do not want want **isprime( )** to be callable from outside **myrandom( )**. In a way we wish to protect it. In such a case we can define **isprime( )** as an inner function.

- Another use of inner functions is in creating decorators. This usage is discussed in Chapter 24.

Communication with Functions

- Communication with functions is done using parameters/arguments passed to it and the value(s) returned from it.

- The way to pass values to a function and return value from it is shown below:

```
def cal_sum(x, y, z) :
    return x + y + z

# pass 10, 20, 30 to cal_sum( ), collect value returned by it
s1 = cal_sum(10, 20, 30)
# pass a, b, c to cal_sum( ), collect value returned by it
a, b, c = 1, 2, 3
s2 = cal_sum(a, b, c)
```

HKOBooks

- **return** statement returns control and value from a function. **return** without an expression returns **None**.
- To return multiple values from a function we can put them into a list/tuple/set/dictionary and then return it.
- Suppose we pass arguments **a**, **b**, **c** to a function and collect them in **x**, **y**, **z**. Changing **x**, **y**, **z** in the function body, does not change **a**, **b**, **c**. Thus a function is always called by value.
- A function can return different types through different return statements.
- A function that reaches end of execution without a **return** statement will always return **None**.

Types of Arguments

- Arguments in a Python function can be of 4 types:

  (a) Positional arguments
  (b) Keyword arguments
  (c) Variable-length positional arguments
  (d) Variable-length keyword arguments

  Positional and keyword arguments are often called 'required' arguments, whereas, variable-length arguments are called 'optional' arguments.

- Positional arguments must be passed in correct positional order. For example, if a function expects an int, float and string to be passed to it, then while calling this function the arguments must be passed in the same order.

```
def fun(i, j, k) :
    print(i + j)
    print(k.upper( ))

fun(10, 3.14, 'Rigmarole')      # correct call
fun('Rigmarole', 3.14, 10)      # error, incorrect order
```

While passing positional arguments, number of arguments passed must match with number of arguments received.

HkgBooks

- Keyword arguments can be passed out of order. Python interpreter uses keywords (variable names) to match the values passed with the arguments used in the function definition.

```
def print_it(i, a, str) :
    print(i, a, str)

print_it(a = 3.14, i = 10, str = 'Sicilian')       # keyword, ok
print_it(str = 'Sicilian', a = 3.14, i = 10)       # keyword, ok
print_it(str = 'Sicilian', i = 10, a = 3.14)       # keyword, ok
print_it(s = 'Sicilian', j = 10, a = 3.14)         # error, keyword name
```

An error is reported in the last call since the variable names in the call and the definition do not match.

- In a call we can use positional as well as keyword arguments. If we do so, the positional arguments must precede keyword arguments.

```
def print_it(i, a, str) :
    print(i, a, str)

print_it(10, a = 3.14, str = 'Ngp')    # ok
print_it(10, str = 'Ngp', a = 3.14)    # ok
print_it(str = 'Ngp', 10, a = 3.14)    # error, positional after keyword
print_it(str = 'Ngp', a = 3.14, 10)    # error, positional after keyword
```

- Sometimes number of positional arguments to be passed to a function is not certain. In such cases, variable-length positional arguments can be received using ***args**.

```
def print_it(*args) :
    print( )
    for var in args :
        print(var, end = ' ')

print_it(10)                                        # 1 arg, ok
print_it(10, 3.14)                                  # 2 args, ok
print_it(10, 3.14,'Sicilian')                       # 3 args, ok
print_it(10, 3.14, 'Sicilian', 'Punekar')           # 4 args, ok
```

KICBOOKS

args used in definition of **print_it( )** is a tuple. * indicates that it will hold all the arguments passed to **print_it( )**. The tuple can be iterated through using a **for** loop.

- Sometimes number of keyword arguments to be passed to a function is not certain. In such cases, variable-length keyword arguments can be received using ****kwargs**.

```
def print_it(**kwargs) :
    print( )
    for name, value in kwargs.items( ) :
        print(name, value, end = ' ')

print_it(a = 10)                              # keyword, ok
print_it(a = 10, b = 3.14)                    # keyword, ok
print_it(a = 10, b = 3.14, s = 'Sicilian')    # keyword, ok
dct = {'Student' : 'Ajay', 'Age' : 23}
print_it(**dct)                               # ok
```

kwargs used in definition of **print_it( )** is a dictionary containing variable names as keys and their values as values. ** indicates that it will hold all the arguments passed to **print_it( )**.

- We can use any other names in place of **args** and **kwargs**. We cannot use more than one **args** and more than one **kwargs** while defining a function.

- If a function is to receive required as well as optional arguments then they must occur in following order:

  - positional arguments
  - variable-length positional arguments
  - keyword arguments
  - variable-length keyword arguments

```
def print_it(i, j, *args, x, y, **kwargs) :
    print( )
    print(i, j, end = ' ')
    for var in args :
        print(var, end = ' ')
    print(x, y, end = ' ')
    for name, value in kwargs.items( ) :
        print(name, value, end = ' ')
```

HkgBooks

```
# nothing goes to args, kwargs
print_it(10, 20, x = 30, y = 40)

# 100, 200 go to args, nothing goes to kwargs
print_it(10, 20, 100, 200, x = 30, y = 40)

# 100, 200 go to args, nothing goes to kwargs
print_it(10, 20, 100, 200, y = 40, x = 30)

# 100, 200 go to args. 'a' : 5, ' b' : 6, 'c' : 7 go to kwargs
print_it(10, 20, 100, 200, x = 30, y = 40, a = 5, b = 6, c = 7)

# error, 30 40 go to args, nothing left for required arguments x, y
print_it(10, 20, 30, 40)
```

- While defining a function default value can be given to arguments. Default value will be used if we do not pass the value for that argument during the call.

```
def fun(a, b = 100, c = 3.14) :
    return a + b + c

w = fun(10)               # passes 10 to a, b is taken as 100, c as 3.14
x = fun(20, 50)           # passes 20, 50 to a, b. c is taken as 3.14
y = fun(30, 60, 6.28)     # passes 30, 60, 6.28 to a, b, c
z = fun(1, c = 3, b = 5)  # passes 1 to a, 5 to b, 3 to c
```

- Note that while defining a function default arguments must follow non-default arguments.

Unpacking Arguments

- Suppose a function is expecting positional arguments and the arguments to be passed are in a list, tuple or set. In such a case we need to unpack the list/tuple/set using * operator before passing it to the function.

```
def print_it(a, b, c, d, e) :
    print(a, b, c, d, e)

lst = [10, 20, 30, 40, 50]
tpl = ('A', 'B', 'C', 'D', 'E')
s = {1, 2, 3, 4, 5}
print_it(*lst)
```

KICBOOKS

```
print_it(*tpl)
print_it(*s)
```

- Suppose a function is expecting keyword arguments and the arguments to be passed are in a dictionary. In such a case we need to unpack the dictionary using ** operator before passing it to the function.

```
def print_it(name = 'Sanjay', marks = 75) :
    print(name, marks)

d = {'name' : 'Anil', 'marks' : 50}
print_it(*d)
print_it(**d)
```

The first call to **print_it( )** passes keys to it, whereas, the second call passes values.

P</> Programs

Problem 13.1

Write a program to receive three integers from keyboard and get their sum and product calculated through a user-defined function **cal_sum_prod( )**.

Program

```
def cal_sum_prod(x, y, z) :
    ss = x + y + z
    pp = x * y * z
    return ss, pp                  # or return(ss, pp)

a = int(input('Enter a: '))
b = int(input('Enter b: '))
c = int(input('Enter c: '))
s, p = cal_sum_prod(a, b, c)
print(s, p)
```

HkgBooks

Output

```
Enter a: 10
Enter b: 20
Enter c: 30
60 6000
```

Tips

- Multiple values can be returned from a function as a tuple.

Problem 13.2

Pangram is a sentence that uses every letter of the alphabet. Write a program that checks whether a given string is pangram or not, through a user-defined function **ispangram( )**.

Program

```
def ispangram(s) :
    alphaset = set('abcdefghijklmnopqrstuvwxyz')
    return alphaset <= set(s.lower( ))
print(ispangram('The quick brown fox jumps over the lazy dog'))
print(ispangram('Crazy Fredrick bought many very exquisite opal
jewels'))
```

Output

```
True
True
```

Tips

- **set( )** converts the string into a set of characters present in the string.
- <= checks whether **alphaset** is a subset of the given string.

Problem 13.3

Write a Python program that accepts a hyphen-separated sequence of words as input and calls a function **convert( )** which converts it into a

HkgBooks

hyphen-separated sequence after sorting them alphabetically. For example, if the input string is

'here-come-the-dots-followed-by-dashes'

then, the converted string should be

'by-come-dashes-dots-followed-here-the'

Program

```
def convert(s1) :
    items = [s for s in s1.split('-')]
    items.sort( )
    s2 = '-'.join(items)
    return s2

s = 'here-come-the-dots-followed-by-dashes'
t = convert(s)
print(t)
```

Output

```
by-come-dashes-dots-followed-here-the
```

Tips

- We have used list comprehension to create a list of words present in the string **s1**.
- The **join( )** method returns a string concatenated with the elements of an iterable. In our case the iterable is the list called **items**.

HkgBooks

Problem 13.4

Write a Python function to create and return a list containing tuples of the form (x, x^2, x^3) for all x between 1 and 20 (both included).

Program

```
def generate_list( ):
    lst = list( )            # or lst = [ ]
    for i in range(1, 11):
        lst.append((i, i ** 2, i ** 3))
    return lst
l = generate_list( )
print(l)
```

Output

```
[(1, 1, 1), (2, 4, 8), (3, 9, 27), (4, 16, 64), (5, 25, 125), (6, 36, 216),
(7, 49, 343), (8, 64, 512), (9, 81, 729), (10, 100, 1000)]
```

Tips

- **range(1, 11)** produces a list of numbers from 1 to 10.
- **append( )** adds a new tuple to the list in each iteration.

Problem 13.5

A palindrome is a word or phrase which reads the same in both directions. Given below are some palindromic strings:

deed
level
Malayalam
Rats live on no evil star
Murder for a jar of red rum

Write a program that defines a function **ispalindrome( )** which checks whether a given string is a palindrome or not. Ignore spaces and case mismatch while checking for palindrome.

Program

```
def ispalindrome(s):
    t = s.lower( )
    left = 0
    right = len(t) - 1

    while right >= left :
        if t[left] == ' ' :
            left += 1
        if t[right] == ' ' :
            right -= 1
```

HkgBooks

```
        if t[left] != t[right]:
            return False
        left += 1
        right -= 1
    return True
print(ispalindrome('Malayalam'))
print(ispalindrome('Rats live on no evil star'))
print(ispalindrome('Murder for a jar of red rum'))
```

Output

```
True
True
True
```

Tips

- Since strings are immutable the string converted to lowercase has to be collected in another string **t**.

HKGBooks

Problem 13.6

Write a program that defines a function **convert( )** that receives a string containing a sequence of whitespace separated words and returns a string after removing all duplicate words and sorting them alphanumerically.

For example, if the string passed to **convert( )** is

s = 'Sakhi was a singer because her mother was a singer, and Sakhi\'s mother was a singer because her father was a singer'

then, the output should be:

```
Sakhi Sakhi's a and because father her mother singer singer, was
```

Program

```
def convert(s) :
    words = [word for word in s.split(' ')]
    return ' '.join(sorted(list(set(words))))
```

```
s = 'I felt happy because I saw the others were happy and because I
knew I should feel happy, but I wasn\'t really happy'
t = convert(s)
print(t)

s = 'Sakhi was a singer because her mother was a singer, and Sakhi\'s
mother was a singer because her father was a singer'
t = convert(s)
print(t)
```

Output

```
I and because but feel felt happy happy, knew others really saw should
the wasn't were
Sakhi Sakhi's a and because father her mother singer singer, was
```

Tips

- **set( )** removes duplicate data automatically.
- **list( )** converts the set into a list.
- **sorted( )** sorts the list data and returns sorted list.
- Sorted data list is converted to a string using a **str** method **join( )**, appending a space at the end of each word, except the last.

Problem 13.7

Write a program that defines a function **count_alphabets_digits( )** that accepts a string and calculates the number of alphabets and digits in it. It should return these values as a dictionary. Call this function for some sample strings.

Program

```
def count_alphabets_digits(s) :
    d={'Digits' : 0, 'Alphabets' : 0}
    for ch in s:
        if ch.isalpha( ) :
            d['Alphabets'] += 1
        elif ch.isdigit( ) :
            d['Digits'] += 1
```

HKGBOOKS

```
        else :
            pass
    return(d)

d = count_alphabets_digits('James Bond 007')
print(d)
d = count_alphabets_digits('Kholi Number 420')
print(d)
```

Output

```
{'Digits': 3, 'Alphabets': 9}
{'Digits': 3, 'Alphabets': 11}
```

Tips

- **pass** doesn't do anything on execution.

Problem 13.8

Write a program that defines a function called **frequency( )** which computes the frequency of words present in a string passed to it. The frequencies should be returned in sorted order by words in the string.

Program

```
def frequency(s) :
    freq = { }
    for word in s.split( ) :
        freq[word] = freq.get(word, 0) + 1
    return freq

sentence = 'It is true for all that that that that \
that that that refers to is not the same that \
that that refers to'
d = frequency(sentence)
words = sorted(d)

for w in words:
  print ('%s:%d' % (w, d[w]))
```

HkgBooks

Output

```
It:1
all:1
for:1
is:2
not:1
refers:2
same:1
that:11
the:1
to:2
true:1
```

Tips

- We did not use **freq[word] = freq[word] + 1** because we have not initialized all word counts for each unique word to 0 to begin with.
- When we use **freq.get(word, 0)**, **get( )** searches the word. If it is not found, the second parameter, i.e. 0 will be returned. Thus, for first call for each unique word, the word count is properly initialized to 0.
- **sorted( )** returns a sorted list of key values in the dictionary.
- **w, d[w]** yields the word and its frequency count stored in the dictionary **d**.

Problem 13.9

Write a program that defines two functions called **create_sent1( )** and **create_sent2( )**. Both receive following 3 lists:

```
subjects = ['He', 'She']
verbs = ['loves', 'hates']
objects = ['TV Serials','Netflix']
```

Both functions should form sentences by picking elements from these lists and return them. Use **for** loops in **create_sent1( )** and list comprehension in **create_sent2( )**.

KICBOOKS

Program

```
def create_sent1(sub, ver, obj) :
    lst = [ ]
    for i in range(len(sub)) :
        for j in range(len(ver)) :
            for k in range(len(obj)) :
                sent = sub[i] + ' ' + ver[j] + ' ' + obj[k]
                lst.append(sent)
    return lst

def create_sent2(sub, ver, obj) :
    return [(s + ' ' + v + ' ' + o) for s in sub for v in ver for o in obj]

subjects = ['He', 'She']
verbs = ['loves', 'hates']
objects = ['TV Serials','Netflix']

lst1 = create_sent1( subjects, verbs, objects)
for l in lst1 :
    print(l)

print( )
lst2 = create_sent2( subjects, verbs, objects)
for l in lst2 :
    print(l)
```

Output

```
He loves TV Serials
He loves Netflix
He hates TV Serials
He hates Netflix
She loves TV Serials
She loves Netflix
She hates TV Serials
She hates Netflix

He loves TV Serials
He loves Netflix
He hates TV Serials
```

HkgBooks

```
He hates Netflix
She loves TV Serials
She loves Netflix
She hates TV Serials
She hates Netflix
```

Exercises

[A] Answer the following questions:

(a) Write a program that defines a function **count_lower_upper( )** that accepts a string and calculates the number of uppercase and lowercase alphabets in it. It should return these values as a dictionary. Call this function for some sample strings.

(b) Write a program that defines a function **compute( )** that calculates the value of n + nn + nnn + nnnn, where n is digit received by the function. Test the function for digits 4 and 7.

(c) Write a program that defines a function **create_array( )** to create and return a 3D array whose dimensions are passed to the function. Also initialize each element of this array to a value passed to the function.

(d) Write a program that defines a function **create_list( )** to create and return a list which is an intersection of two lists passed to it.

(e) Write a program that defines a function **sanitize_list( )** to remove all duplicate entries from the list that it receives.

(f) Which of the calls to **print_it( )** in the following program will report errors.

```
def print_it(i, a, s, *args) :
    print( )
    print(i, a, s, end = ' ')
    for var in args :
        print(var, end = ' ')

print_it(10, 3.14)
print_it(20, s = 'Hi', a = 6.28)
print_it(a = 6.28, s = 'Hello', i = 30)
print_it(40, 2.35, 'Nag', 'Mum', 10)
```

HkgBooks

(g) Which of the calls to **fun( )** in the following program will report errors.

```
def fun(a, *args, s = '!') :
    print(a, s)
    for i in args :
        print(i, s)

fun(10)
fun(10, 20)
fun(10, 20, 30)
fun(10, 20, 30, 40, s = '+')
```

[B] Attempt the following questions:

(a) What is being passed to function **fun( )** in the following code?

```
int a = 20
lst = [10, 20, 30, 40, 50]
fun(a, lst)
```

(b) Which of the following are valid **return** statements?

```
return (a, b, c)
return a + b + c
return a, b, c
```

(c) What will be the output of the following program?

```
def fun( ) :
    print('First avatar')
fun( )
def fun( ) :
    print('New avatar')
fun( )
```

(d) How will you define a function containing three **return** statements, each returning a different type of value?

(e) Can function definitions be nested? If yes, why would you want to do so?

(f) How will you call **print_it( )** to print elements of **tpl**?

```
def print_it(a, b, c, d, e) :
    print(a, b, c, d, e)
tpl = ('A', 'B', 'C', 'D', 'E')
```

HkgBooks