CSX3001/ITX3001

FUNDAMENTALS OF COMPUTER PROGRAMMING

CLASS 11 RECURSIVE FUNCTIONS, MODULES IN PYTHON

PYTHON

A FUNCTION CAN CALL OTHER FUNCTIONS

We know that in Python a function can call other functions. Let’s look at the following example.

```
def printName(name):
    print(f'Dear, {name}')

def printInfo(name, homeaddr):
    printName(name)
    print(f'Your home address is {homeaddr}')

def readInfo():
    name = input('Enter your name: ')
    addr = input('Enter your hometown: ')
    printInfo(name, addr)

#main body
print('Welcome to our service.')
readInfo()
```

Run the above code and see the result. As you can see that in the main body, a function, namely readInfo(), is called. In readInfo() function it calls printInfo() function, which again calls printName() function.

Nonetheless, in most used programing languages, there is a function that calls itself. That is known as a recursive function.

WHAT IS A RECURSIVE FUNCTION?

When a function call itself, it is referred to as a recursion. Recursion works like loop but sometimes it makes more sense to use recursion than loop. You can convert any loop to recursion.

Here is how recursion works. A recursive function calls itself. As you'd imagine such a process would repeat indefinitely if it is not stopped by some conditions. This condition is known as a base condition. A base condition is a must in every recursive function otherwise such recursive function will continue to execute forever like an infinite loop.

Overview of how recursive function works:

1. Recursive function is called by some external code.
2. If a base condition is met, then the function does pre-defined tasks and exits.
3. Otherwise, a function does other tasks and then call itself to continue recursion. Here is an example of a recursive function that is used to calculate factorial.

Note: Factorial is denoted by “!” sign. For example,

```
4! = 4 * 3 * 2 * 1
2! = 2 * 1
0! = 1
```

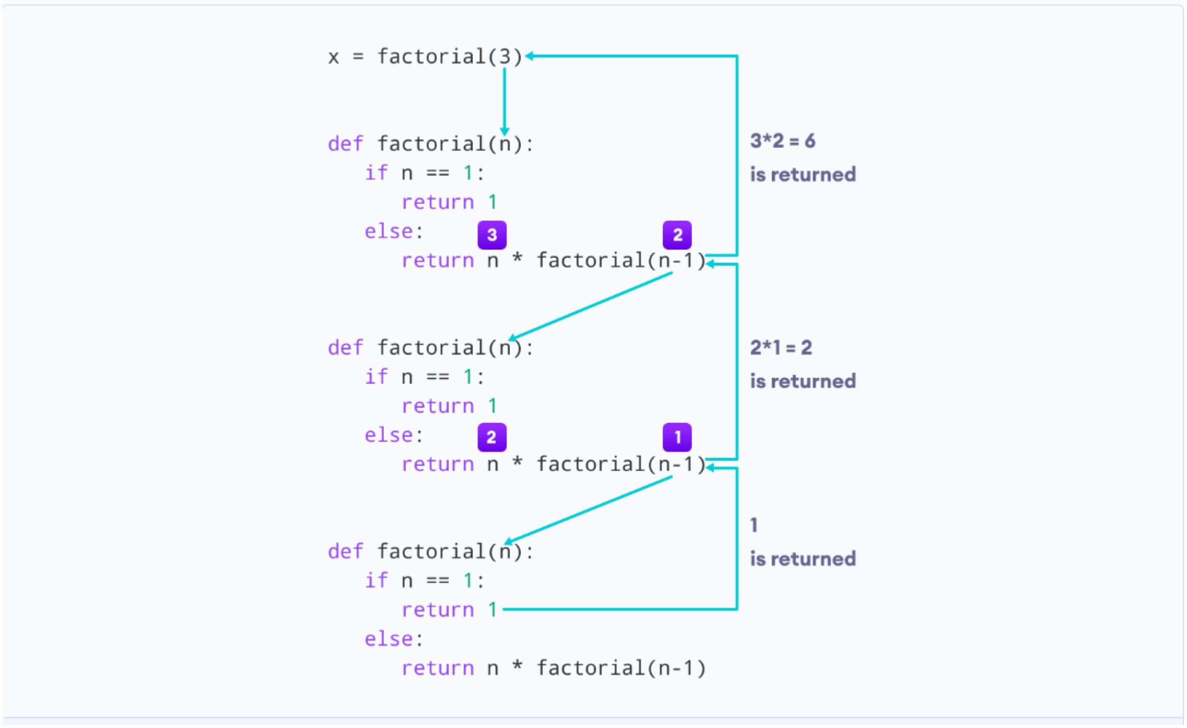
```
# A typical loop used to find a factorial value
  n = 4
  if n == 0:
      fac = 1
  else:
      fac = 1
      for i in range(1, n+1):
          fac = fac * i
  print(fac)

# ======== A recursive function used to find a factorial value
  def fac(n):
      if n == 0:
          return 1
      else:
          return n * fac(n-1)
  n = 4
  print(fac(n))
```

Our recursion ends when the number (*n*) reduces to 1. This is called the base condition. Otherwise, in the above code, this function calls itself with a passing parameter, *n*-1, and multiplies a result from that function call with *n*. This recursion will end when the base condition is met.

This recursive call can be explained in the following steps.

```
fac(3)              # 1st call with 4
3 * fac(2)          # 2nd call with 3
3 * 2 * fac(1)      # 3rd call with 2
3 * 2 * 1           # return from 3th call as number=1
3 * 2               # return from 2rd call
6                   # return from 1nd call
```



```
x = factorial(3)

def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)

def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)

def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

Every recursive function must have a base condition that stops the recursion or else the function calls itself infinitely.

The Python interpreter limits the depths of recursion to help avoid infinite recursions, resulting in stack overflows.

By default, the maximum depth of recursion is 1000. If the limit is crossed, it results in RecursionError. Note: this limit can be changed by calling .setrecursionlimit() in sys module.

Advantages of Recursion

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

RECURSIVE EXERCISES

1) Write a recursive function, namely printHarry(n), that prints 'Harry' *n* times.

2) Write a recursive function, namely list_sum(num_list), that calculates a sum of all numbers in the list.

The given code finds the sum of a non-negative integer

```
n = 34565
total = 0
while n > 10:
    remainder = n % 10
    n = int(n / 10)
    total += remainder
else:
    total += n
print(total)
```

3) Write a recursive function, namely sumdigits(), to calculate a sum of positive integer's digits. For example,

   sumdigits(345) will return 12 (3 + 4 + 5)

4) Write a recursive function, namely sum_series(n) that calculates a sum of positive integers of n+(n-2)+(n-4)... (until n-x <= 0).

5) Write a recursive function, namely power(x,y), that calculates a value of 'x' to the power 'y'.

WHAT ARE MODULES IN PYTHON?

You can import existing Python modules in which each module contains a set of functions that you can call (using ***import*** statement to import Python modules). For example, The math module is a standard module in Python and is always available. To use mathematical functions under this module, you must import the module using ***import math*** statement.

```
import math

#to find a square root, call a function available in math
#module using dot '.' operator, for example

result = math.sqrt(16)
print(result)
```

Each individual Python module contains different functions, visit the following link for examples of functions in module math.
https://www.programiz.com/python-programming/modules/math

However, it is also possible to write your own module in Python. You can define your most used functions in a module and import it, instead of copying their definitions into different programs.

Let us create a module. Type the following and save it as MyCal.py.

```
def add(a, b):
   result = a + b
   return result

def subtract(a,b):
    result = a - b
    return result
```

Here, we have defined functions add() and subtract() inside a module named *MyCal*. These functions take in two numbers and return the result.

HOW TO IMPORT A MODULE IN PYTHON?

We can import the definitions inside a module to another module or the interactive interpreter in Python.

We use the ***import*** keyword to do this. To import our previously defined module *MyCal*, write the following code in your Python program:

```
import MyCal
```

This statement does not enter the names of the functions defined in *MyCal* directly in the current symbol table. It only enters the module named *MyCal* there.

With a module name we can access the function using the dot '.' operator.

```
import MyCal

result = MyCal.add(4,5)
print(result)

result = MyCal.subtract(10,6)
print(result)
```

Note that Python has a ton of standard modules available.

You can check out the full list of Python standard modules and what they are for. These files are in the Lib directory inside the location where you installed Python.

Standard modules can be imported the same way as we import our user-defined modules.

MODULE EXERCISE(S)

6) Write a module namely myinfo (create and save myinfo.py). The module has two functions, namely printName(name) and printYoB(age).

   printName(name) takes your name and prints "Dear {name}, welcome to Python Module exercise."

   printYoB(age) takes your current age and prints your year of birth as "You were highly likely born in Year {2020 – age + 1}".

   Complete the following instructions in a main body of your code:

   - Import the written module
   - Take your name and your age as inputs, and store them in two variables, namely *name* and *age*, respectively.
   - Call printName() and pass *name* as an input argument
   - Call printYoB() and pass *age* as an input argument

   E.g.

```
Enter your name: Harry
Enter your age: 10
Dear Harry, welcome to Python Module exercise. You were
highly likely born in Year 2011.
```

7) Write a module namely simplemath that contains four functions as follows:
   - fac(n), which finds factorial of n (using recursive function)
   - avglist(numlist), which finds an average value of a list of integers
   - sumofoddpos (numlist), this recursive function finds a sum of all integers in odd position. E.g., with numlist = [10,2,30,4,50,6] a result must be 10 + 30 + 50 (a position of an item in the list, not an index)
   - sumofevenpos (numlist), this recursive function finds a sum of all integers in odd position. E.g., with numlist = [1,20,3,40,5,60] a result must be 20 + 40 + 60

   Write a code to import *simplemath* module and test each function.

ASSIGNMENT

1. Write a module, namely ***MyOwnFunctions***, which contains the following functions:
    a. ***SplitType(numlist)*** which takes a list of numbers (integer or float) and returns a list of only integers and a list of only floats from a numlist.
    b. ***ListofOdd(numlist)*** which takes a list of integers and returns a list of only odd integers. Note that this function must be a recursive function.
    c. ***KeepTwoDup(numlist)*** which takes a list of integers and returns a list of unique integers and duplicated integers that a maximum number of duplications is two. For example
    numlist = [1,2,2,2,3,4,5,5,5,1,2,3,3]
    outlist = [1, 2, 2, 3, 4, 5, 5, 1, 3]

    d. Write a complete Python code to import *MyOwnFunctions* and test each individual function accordingly.

    Hints: for ListofOdd(numlist), the below function returns all integers from the original numlist.

```
def ReturnAllNum(numlist):
    if len(numlist) == 1:
        outlist.append(numlist[0])
        return outlist
    else:
        outlist.append(numlist[0])
        return ReturnAllNum(numlist[1:]) # Recursive Call
```

    For KeepTwoDup(), modify RemoveDuplicate() by adding seen2 = set() and add necessary if-else condition(s).