More about Functions

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Functions – Recap

- Functions are defined - to use them they are called - when called their body is executed in a function frame
- Functions have parameters, for calling a function we have to provide arguments - which must match the parameters
- With functions, our program is a sequence of function definitions and a sequence of statements (of main program) - computation is mostly done in functions, main program coordinates
- Functions are essential for solving large problems - provide a method of divide-and-conquer and modularizing the program

Functions Recap

- All variables defined in a function (including parameters) are local variables - available only inside the function body (when the function is executing)
- Variables in main program can be accessed / modified in functions - though strongly discouraged
- Scope of variables: where the variable is visible; local variable's scope is the function, global var scope is whole program

Functions can call Functions

- In maths we have learned that we can compose functions, e.g.

  $g(x, y) = f1(x) + f2(y)$

  $g(x) = f1\ (f2(x))$

  Say, f1 = compute square, f2 = compute sq root

- In python, functions can also call functions, allowing very flexible functional composition
- Both the above type of composition are allowed
- We will consider all functions are defined at the program level (later may discuss advanced concept of function defined within a function)
- Functions at the program level can call each other
- A function can also call itself - recursion - we will discuss it later

Functional Composition

```
def f1(x):
    return x*x

def f2(x):
    return x**(1/2)

def g(x,y):
    val = f1(x) + f2(y)
    return val

# main program
a, b = 3, 4
val = g(a,b)
print(val)
```

```
def f1(x):
    return x*x

def f2(x):
    return x**(1/2)

def g(x):
    val = f1(f2(x))
    return val

# main program
a, b = 3, 4
val = g(b)
print(val)
```

Functional Composition Example

```python
# A function to calculate area of floor
def calculate_area(length, width):
    area = length * width
    return area

# A function to calculate total cost
def calculate_cost(area, price_per_square_foot):
    cost = area * price_per_square_foot
    return cost

# Program to calculate cost of carpeting a rectangular room
len = 10
wid = 20
per_unit_cost = 650
ans = calculate_cost(calculate_area(len, wid), per_unit_cost)
print("Cost of carpet (Rs): ", ans)
```

Functional Composition – Compute nCr

```
# Fn to compute factorial
def factorial(n):
    fact = 1
    for i in range(1,n+1):
        fact = fact*i
    return fact

# Fn to compute nCr - calls factorial
def combination(n,r):
    C = factorial(n)/(factorial(r)*factorial(n-r))
    return C

# Main Program - to compute the nCr
n = int(input("Enter the value of n: "))
r = int(input("Enter the value of r: "))
print("The value of nCr is: ",combination(n,r))
```

Quiz – Single Correct

Choose the correct statement that completes the code for finding the distance between two points on the plane - (x_1,y_1) and (x_2,y_2).

```
A.  distance = g(f(h(x1,x2)) + f(h(y1,y2)))
B.  distance = f(g(h(x1,x2)) - g(h(y1,y2)))
C.  distance = f(g(h(x1,y1)) * g(h(x2,y2)))
D.  distance = g(h(f(x1,x2)) + h(f(y1,y2)))
```

```
def f(a):
    return a*a
def g(a):
    return a**0.5
def h(a,b):
    return a-b

x1 = 1
y1 = 2
x2 = 4
y2 = 6

# Enter Code Here
print(distance)
```

Quiz – Single Correct

Choose the correct statement that completes the code for finding the distance between two points on the plane - (x_1,y_1) and (x_2,y_2).

```
A.   distance = g(f(h(x1,x2)) + f(h(y1,y2)))
B.   distance = f(g(h(x1,x2)) - g(h(y1,y2)))
C.   distance = f(g(h(x1,y1)) * g(h(x2,y2)))
D.   distance = g(h(f(x1,x2)) + h(f(y1,y2)))
```

```
def f(a):
    return a*a
def g(a):
    return a**0.5
def h(a,b):
    return a-b

x1 = 1
y1 = 2
x2 = 4
y2 = 6

# Enter Code Here
print(distance)
```

First we find x1-x2 and y1-y2, then we square each of them, add them up and finally take square root of the result to obtain the distance.

Keyword Arguments

- Another way to pass arguments in python - keyword arguments
- A function:

```
def fn(var1, var2, z):...
```

- Can be called by:

```
fn(var1=value1, var2=value2, z=val3)
```

- I.e. names of parms are used, and argument value explicitly tied to the name
- The parameter name should be exactly the same as in definition
- Order is now not important (as param-arg mapping is explicit)

Arguments – positional and keyword

```
def cost (a, b, c):
    totcost = a*c
    print("Item, and total cost are: ", b, totcost)
```

```
# Call using positional arguments
item = 5
qty = 3
unit = 200
totcost = cost(qty, item, unit)
# qty assigned to a, item assigned
to b, unit to c)
```

```
# Call using keyword arguments
item = 5
qty = 3
unit = 200
totcost = cost(b=qty, a=item,
c=unit)
# order of args not important
```

Mixing of Argument Types

- Possible to have some positional and some keyword args in a call
- All positional args must come first, then the keyword args
- I.e. there cannot be any positional args after a keyword arg

```
# Call using positional & keyword arguments
item = 5
qty = 3
unit = 200
totcost = cost(qty, c = unit, b = item )


# qty assigned to a, item assigned to b, unit assigned to c)
```

Default Parameters

- When defining a function, can assign values to some of the parameters also in the function head
- These values become default values - if the call does not provide an arg for it, the default value is used
- Allows calling function to not specify args for all parms (i.e. args for all parameters to be given is not fully true)
- Note: Any default parameter should always be after the non default parameters
- Eg. a function:

```
def my_fn(a, b, c=10):
    return(a*b*c)
# Calling from main
my_fn(1, 2, 3)  # returns 6
my_fn(1, 2) # returns 20
```

Quiz

Consider the given code that adds should return sum of two numbers. Re-write Line 1 such that the code produces 10 as the output.

Note: Do not use any blank spaces (‘ ’) in use answer except for the one between ‘def’ and the function name.

```
1  def add(a,b):
2        return a + b
3  a = 4
4  sum = add(a)
5  print(sum)
```

```
Output : 10
```

Quiz(Solution)

Consider the given code that adds should return sum of two numbers. Re-write Line 1 such that the code produces 10 as the output.

```
1   def add(a,b=6):
2         return a + b
3   a = 4
4   sum = add(a)
5   print(sum)
```

```
Output : 10
```

Solution- def add(a,b=6):

Doc Strings

- Doc strings are attached with functions (and some other objects)
- They are not executed, but are recognized
- For a function, there are some automatically defined methods, and the __doc__ for a function will give the doc string
- Doc strings helps to describe the job of the function, specify required parameter types and also specify the return type of the function.
- The docstrings are declared using ”’triple single quotes”’ or “””triple double quotes””” just below the function declaration
- Desirable: All functions should have a docstring describing what the function is doing (not how or the logic)

Functions with variable parameters

- This is an advanced topic - we will not cover it
- Functions can have variable number of parameters
- Requires arguments to be packed and passed, and then unpacked at the function
- ….

Developing Programs – Top-Down Approach

- For writing a program, one approach is top-down development
- Start by writing the program (approach) for solving the problem
- Whenever you need some value for which a separate computation is needed - call a function, and define a dummy function
- Continue developing the main program
- Can run the main program with dummy functions
- Then write code for implementing the functions - can do it incrementally

Example Problem: Given a number, find if it is prime, if not, find its prime factors

Approach

1. Get the number
2. Check if it is prime
3. Else
4. [First repeatedly divide it by 2 till an odd number]
5. Determine the prime factors of this odd number

Can include step 4 in determining of prime factors also

Will have main, and isprime(), and primefactors()

Can have isprime() dummy return True / False to try

Then work out code for prime

Work out code for primefactors

Code

```
# Main Program
x=int(input("Give an Integer:"))
print("x: ", x)
if isprime(x):
    print(x, " is a prime")
else:
    #First repeatedly divide by 2 till it
is odd no
    print("Prime factors are")
    if x%2==0:
        print("2 is a factor")
        while x%2 == 0:
            x = x // 2
    # Get prime factors of this odd no
    primefactors(x)
```

```
# Two functions - final
def isprime(i):
    j = 2
    isprime = True
    while(j <= i**(1/2)):
        if (i%j) == 0:
            isprime=False
        j = j + 1
    return isprime

def primefactors(x):
    i = 3
    while (i<= x):
        if isprime(i):
            if x%i == 0:
                print(i,"is a factor")
                while x%i == 0:
                    x = x//i
        i = i + 2
```

Summary

- Functions are a powerful way to break the problem into smaller problems, write functions for smaller ones, and then combine them into a solution
- Functions can have parameters; functions are called with arguments - values of args assigned to parms
- All vars in a function (incl parms) are local - they are accessible only inside the function
- There is no name conflict between local vars and global vars - i.e. var x defined in main, and var x defined in a fn are completely different - in fn x will refer to local var, in main it will refer to its x
- Functions can call functions

Summary

- Functions can be called with positional arguments - # of args must be same as # of parms, args assigned to parms in order
- Functions can also be called with keyword args - then the order of args does not matter
- Positional and keyword args can be combined - all positional must come before any keyword arg
- A function can call functions - allows composition

- You are now empowered to solve a range of problems - you know the main language constructs, and functions which help in problem solving through divide-and-conquer
- Assignment 1 to be given - solving problems through programming

Functions as Parameters/Arguments

- A function can have some parameters that are functions, i.e. the calling program has to pass functions as arguments for these parms
- Such functions which are passed functions are called higher order functions
- Allows writing functions to do general computations using any function (i.e. at code writing time, you dont know what the function does - you just know that it is a function)
- These functions can use the parameter which is a function as a function in its code - the passed function will be executed
- Interesting things can be done when functions are passed - for now we will keep it simple - the higher order function can just call the function

Functions as Parameters

Defining a Higer Order Function

- Defining - suppose one function parameter and one value

```
def hof (f, x):
    ...
    f(x)
    ...
```

- Just like a regular function definition - first parameter is a function - in the body the function can be called

Calling a HOF

- Calling a HOF function with a function argument

```
    def g(x):
    # code for f
#main code
X = 0
hof(g, x)
```

- Calling hof - the first argument is a function

Functions as Parameters

- Note that passing a value computed from a function is very different from passing a function
- If you want to pass the value computed using the value f(x), you will pass f(x) as a value parameter

```
hof2(f(x))
```

- Here f(x) is computed and value passed - different function required for this
- Passing the function - call with just f , i.e. without the ()

```
hof(f, x)
```

-

Example

Want to compute value of a given function from integer range x1, to x2, with increments of x

```
def fn_values(f, x0, x1, d):
    for i in range(x0, x1, d):
        y = f(i)
        print("for x: ", i, ", y is: ", y)
```

```
def sq (x):
    return x*x
def cu(x):
    return x*x
print("Computing 1st function")
fn_values(sq, 1, 5, 1)
print("\nComputing 2n  function")
fn_values(cu, 2, 6, 1)
```

Importing Functions and Using Them

- Functions are reusable - they get what they need for computation through the parameters and just return some value (assuming no side effect or use of global variables values)
- It makes sense to have standard functions written for commonly used functions (e.g. sin, cos, log, …) and make it available
  - Note in programing if you want to use sin(x) for some x in a program to solve some problem, you have to *compute* sin(x)
- With such functions, the programming language effectively has many more primitives / operations and not just *, **, //, …
- For math functions, python provides a module which has a range of functions defined in it

Python's math module

- These functions are defined in a module **`math`** - to use these functions you have to import the module using:

```
import math # is a statement in your prog
```

- Importing a module effectively makes all the functions defined in math available to you in your program - you use a function by:

```
math.log(x)
```

- You can see all the functions (and constants) available in math using the **`dir`** function - a standard function provided by python
- You can find the type of a variable using the **`type`** function
- You can find out about a function (or a statement) using help on the python terminal
- Illustrate dir, help, using functions of math…

Using math functions

- For using anything in math, you have to use the dot notation
- E.g. to do sin(x)**2 + cos(x)**2 you have to write

  math.sin(x)**2 + math.cos(x)**2

- You can avoid this by targeted importing of the functions from the math module using:

```
from math import sin, cos
```

- Now you can use sin, cos without the dot

Quiz

What will be the output of this program:

```
import math
def f1(f, n):
    return f(n)
def f2(f,n):
    return f*n
x = f1(math.sqrt,9)
y = f2(math.sqrt(9),9)
print(x+y)
```

Quiz

- 6 will pass the sqrt fn and 9, and f1 will compute 3
- 7 will pas 3 and 9 and f2 will compute 27
- Final answer 30.0

```
1. import math
2. def f1(f, n):
3.     return f(n)
4. def f2(f,n):
5.     return f*n
6. x = f1(math.sqrt,9)
7. y = f2(math.sqrt(9),9)
8. print(x+y)
```

Modules

- Python has many built in modules like math - they provide many useful functions which can be used by module users
- Programmers can also create modules - and use them in their own program or share them with others
- We will discuss the issue of modules, packages, etc later, and what importing does

Order of Evaluation

- With functions now an expression can have (i) values, (ii) variables, (iii) operations, and (iv) functions
- We have seen precedence of operators - with functions, functions are higher than all operations - i.e. functions are evaluated first and then the rest of the operations are performed (in order)
- A functions parameters can also be functions or expressions - parameters are at same level, so evaluated left to right

Summary

- We can define a function which has functions as parameters - these parameters are used as functions in the function body
- Allows us to write higher order functions - those that work on functions
- Function parameters can be called in the function
- Passing the function parameter - just the function name, without ()
- We can import the math module in our program - then a lot of math functions become available to our program - these are like user defined functions - can pass them as parameters

Announcements

- This is a good point to pause and solidify our learning
- Next class we will spend some time to clarify any point not clear - you can post it on GC before the class, or come to class with it
-
- Assignment 1 will be released - you will get 2 weeks
- To be done on VS code - Sat workshop on it
-
- Next mon, we will probably have a quiz in class - all students have to come in person - details will be announced on GC (only students with proper medical with the acad dept will be excused)

Minute Paper