Functions – and Program Structuring

INDRAPRASTHA INSTITUTE *of*
INFORMATION TECHNOLOGY
DELHI

Recap

- We have seen program as a monolith sequence of statements
- Main types of statements
  - Assignment - var = expression
  - Conditional - if-then-else, if-then, if-elifs
  - Iteration - for loop, while loop
- These statements are sufficient to compute anything computable
- For a large program, or a complex problem
  - One monolith seq of statements is hard to construct or debug
  - Having only the above statements makes it harder
- Functions provide an answer to both of these
  - Allows us to build new and more powerful "constructs" from the basic language constructs, which we can use in our code
  - Allows code to be broken into pieces

Recap

You learn programming by practice

Always remember that

The more you practice, the better you will get

There is no short cut

Functions

- In math, we have functions like:

  Z = f(x,y)

- After defining a function, we can use it in other functions
- In python, we can define very general functions, and use them
- Like in math, functions may have parameters, and to compute a function, values of parameters have to be provided
- Function is a unit of computation – which can be invoked from different places, i.e. used wherever we want
- With functions, a python program is a set of function declarations, and a “main program” which calls / uses these functions
- Lets show it by example

Python Functions: Example

```
# defining a fn sq
def square(x):
    return x*x

# defining a fn cube
def cube(y):
    return y*y*y

# Main program
a, b = 2, 4
c = square(a) + cube(b)
print("Val of c: ", c)
```

- Two functions defined - each has one parameter
- Code of function definition specifies the computation the fn does
- Function can return some value
- To use the function – it is called, value of parameter is provided
- On call – parameter gets value, body of function executed; value returned (and can be used)

Defining and Calling a function

- We need two basic capabilities - defining a function and calling a defined function
- Defining a function is done by def

```
def fn_name(parm-list):
    <fn-body>
```

- Parameters are optional; parameters are available for use in body
- The function execution terminates when it executes a return statement, or its body completes

Defining and Calling a function...

- Defining a function just defines it, to execute we must call it
- Statement to call a function: just the function name with parameters:

```
fn_name(arguments)
```

- If function does not have any arguments, it must be called with () - this tells the interpreter that this is not a variable but a function call

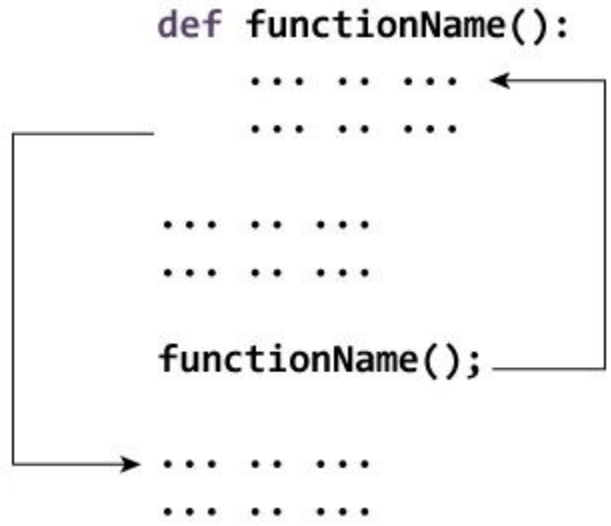
```
fn_name()
```

- If function has parameters, arguments need to be provided for all the parameters - provide value of the parameter for fn execution

Defining and Calling: flow of control

- Program is a sequence of statements being executed by interpreter
- Function definition is a definition - not an executable statement
- Function call is an executable statement
- On encountering a function call statement, to execute it:
    - Interpreter jumps to function definition
    - Parameters are assigned values that corresponding arguments in the calling statement have
    - Body of the function (a sequence of statements) is executed
    - Upon completion of the function, the control returns to the calling stmt
    - Return value, if any, is used where the function was called

Flow of control – diagram (programiz)



```
def functionName():
    ... .. ...
    ... .. ...

... .. ...
... .. ...

functionName();

... .. ...
... .. ...
```

Executing a Program with Functions

A general program structure:

```
def fn1 ():
    body
def fn2 ():
    body
def fn3(params):
    body
# Main program
Stmt-block
# includes some call stmts
```

When interpreter gets this program

- On function definitions, it records some information; body is not executed
- Starts execution from the first stmt in the stmt block of the main program
- On a call statement, control is transferred to the function; function starts executing
- On return statement in the function, goes back to the call stmt (in the main program)
- Execution continues in the main program
- Note: Function definition must be before the function call stmt is executed. Otherwise results in error.

Return statement

- A function can use in its body a special statement:

```
return <expression> # expression is optional
```

- Return statement serves two purposes
  - Terminates the execution of the function and returns the control back to where the function was called
  - Returns a value to the caller
- A function execution can also terminate when its body finishes
  - Like having a return statement as the last statement
- If some value specified in return - that is provided at calling point
- Otherwise the return value is treated as **`None`** (a special value)

Note: In Python, functions can return multiple values. Just write each value/expression after return, separated by commas.

Quiz – Single Correct

Order in which names of colors are printed when the program is executed?

A. Red, Yellow, Blue, Green
B. Red, Green, Yellow, Blue
C. Yellow, Green, Blue, Red
D. Red, Yellow, Green, Blue

```
print("Red")
def f(a, b):
    s = a*b
    print("Green")
    return s

print("Yellow")
num1 = 10
num2 = 5
ans = f(10, 5)
print("Blue")
```

Quiz – Single Correct

Order in which names of colors are printed when the program is executed?

A. Red, Yellow, Blue, Green

B. Red, Green, Yellow, Blue

C. Yellow, Green, Blue, Red

D. Red, Yellow, Green, Blue

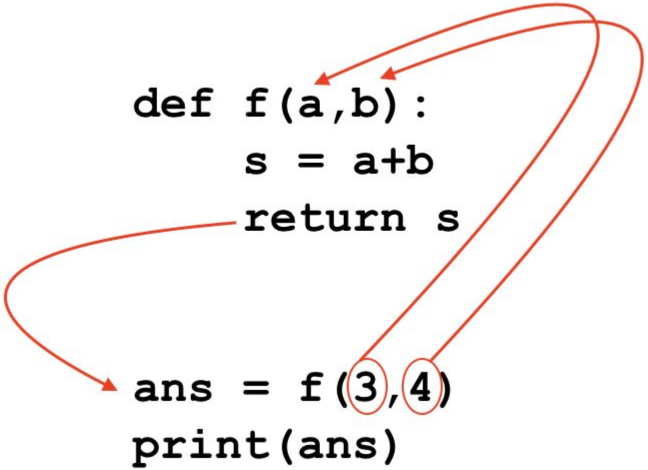
```
print("Red")#1
def f(a, b): #6
    s = a*b #7
    print("Green") #8
    return s #9

print("Yellow")#2
num1 = 10 #3
num2 = 5 #4
ans = f(10, 5) #5
print("Blue") #10
```

Argument Passing

- A function definition may have parameters (or not) - these are available inside the function for use
- Call to a function has arguments (or not)
- When a function is called, argument values are assigned to the parameters
- Positional arguments (also called required arguments) - arguments are assigned to parameters in order
  - Must have same number of arguments for calling
  - i-th argument value is assigned to i-th parameter
- When a function is called, interpreter checks if the # of args is same as # of parms
  - If number of arguments is not same, error

Argument passing example



```
def f(a,b):
    s = a+b
    return s


ans = f(3,4)
print(ans)
```

- The values 3 and 4 are passed as arguments for function `f`.
- The arguments values are copied to function parameters `a` and `b`.
- `a` and `b` are used for computing value of `s`.
- `s` is returned by the function and assigned to variable ans.
- The variable ans now holds the value 7 and is printed.

Argument Passing ...

- Arguments can be pass by value or pass by reference
    - Pass by value - the value of arg is assigned to the parm
    - Pass by reference - a reference to the arg is assigned to the parm - in this case changes made by function can be reflected in the caller
    - Python uses pass by value, but in some cases, this value is a reference - we will discuss it later
- Complex objects can also be passed as arguments

Feedback – Labs

Feedbacks are ANONYMOUS - your email id is NOT recorded

Green: Have done all lab problems myself

Yellow: Have done more than half the problems myself

Red: Have done less than half problems myself

Feedback

Green: Comfortable with material covered so far in IP

Yellow: Somewhat - need to practice more and revise

Red: Uncomfortable

Local and Global Variables

- All vars defined in the function are local - they exist when function is executing
- Parameters of a function are also local variables
- As vars in a function are local to a function - many fns can have same var names with no conflict
- Main program cannot use/access local variables of functions (they don't exist when control is in the main program)
- But the variables of main can be accessed within a function - these are called global variables, which can be accessed in any function
- Accessing/modifying global variables within a function is to be strictly avoided (only to be used rarely)
- Python requires any global variable to be used in a function to be explicitly declared as global, e.g. within the body of the function

```
global X # there must be a variable in main program named X
# X now will refer to the global variable X
```

Scoping of Variables

- Variables keep values; a variable is defined when we assign some value to it
- Some languages require variables to be declared before they can be assigned anything
- Scope of a variable - where the variable can be accessed
- Global variables - those defined outside any function - they are potentially accessible from anywhere in the program, including fns
- Local variables - those defined in a function, including the parameters - only accessible within the function
- To access global variables from within a function, need to specify explicitly that the variable is global - python assumes all variables in a function to be local
- Avoid the use of global variables

Quiz – Single Correct

What will be the output of the code ?

A. 1
B. 2
C. 3
D. 4

```
x = 1
def f(x):
    y = x
    print(y+1)

x = 2
f(3)
```

Quiz – Single Correct

What will be the output of the code ?

```
x = 1
def f(x):
    y = x
    print(y+1)

x = 2
f(3)
```

A. 1
B. 2
C. 3
D. 4

The value 3 passed as argument to function f is copied to parameter x. Then y is assigned the value of local variable x (i.e. 3). Thus the final output is y+1 (i.e. 4)

Main program with functions

- With functions, most of the computation should be done in the functions; the main program should have minimal computations
- A common way to structure the overall code
  - Define functions for units of computation with clean interfaces - i.e. a few parameters and some return value
  - The main program is used mostly for: getting inputs, calling functions to do the processing, and then printing the final results
  - In complex programs, you may even have functions for input/output
- Now whenever you write a program, use functions liberally to do most of the computation
- Lets see an example - program to compute factorial

Example – Factorial

```
# A function to take in a number and print its factorial
def factorial(n):
    fact = 1
    for i in range(1,n+1):
        fact = fact*i
    return fact

# Main Program
n = int(input("Enter an integer: "))
if n<=0:
    print("Number is <= 0")
else:
    fact = factorial(n)
    print("Factorial is: ", fact)
```

Visualizing Execution using Pythontutor

- Helps in visualizing what is happening in the program
- Aids understanding of the working / running
- Can use for debugging
- For small programs - learning a construct or a new feature
- Let us run this program

```
def fn(x,y):
    c = x+y
    return c
a = 5
b = 7
d = fn(a, b)
print(d)
```

And the factorial program

Examples

```
# Fn to find period of pendulum
def pendulum_period(len):
    g = 9.8
    pi = 3.14
    period = 2*pi*((len/g) ** 0.5)
    return period

# Main Program
l = 2.4
ans = pendulum_period(l)
print(ans)
```

```
# Fn to compute simple interest
def simple_interest(principal,
rate, time):
    interest = principal * rate
* time
    return interest

# Main Program
P = 1000
R = 0.05 # Rate of interest 5%
T = 5

SI = simple_interest(P, R, T)
print(SI)
```

Example – HCF

```
def hcf(a,b):
    if a > b:
        smaller = b
    else:
        smaller = a
    for i in range(1, smaller + 1):
        if a % i == 0 and b % i == 0:
            hcf = i
    return hcf

# Main Program
a = 54
b = 12
print("The H.C.F. of", a, "and", b, "is", hcf(a,b))
```

Example – Roots of Quadratic Polynomial

```
# Calculates the solutions to the quadratic a*x**2 + b*x + c = 0
def quadratic_roots(a, b, c):
    # Calculate the discriminant
    d = (b**2) - (4*a*c)
    # Test if discriminant is negative
    if d < 0:
        return None
    else:
        # Calculate the two roots
        x1 = (-b + d**0.5)/(2*a)
        x2 = (-b - d**0.5)/(2*a)
        return (x1, x2)

# Main Program
a, b, c = 1, -5, 6
print("The solutions are:", quadratic_roots(a, b, c))
```

Importance of Functions

- Functions are a powerful tool for writing large programs
    - **Divide and conquer** - allows the large programming problem to be divided into smaller problems, with functions written for solving sub-problems, then combined to solve the problem
    - **Abstraction** - encapsulate a computation to be used anywhere by just using the function name; don’t have to understand function logic for using (it may be written by someone else)
    - **Reusability** - the same function can be called from many places, i.e. the function code is being reused many times
    - **Modularity** - with function, a program is a set of functions (modules) multiple of these are connected together for building a solution
- Functions are the oldest method in programming languages for providing modularity, abstraction, etc
    - Even the earliest languages provided this abstraction

Defining Functions – Some Practices

- A function must have an expressive name which represents what the function is doing
    - Generally names start with lower case letter
- Should have a clean and simple interface - with few parameters
- Should be computing something that can be easily stated in a simple sentence
- Should not have any side effects - i.e. caller only gets returned values, no other changes in any vars in caller or main program
- Must have a comment - which states succinctly ***what*** the program is doing (not its logic) - is also a test of whether the function has a clean abstraction / purpose
- Naming standards - PEP 8 has conventions followed widely

More about Using Fns for Modularity

- Use of functions in the program promotes modularity and code reusability.
- Modular programming emphasise on subdividing a computer program into separate sub-programs (functions) to increase the maintainability, readability of the code and to make it easier to introduce any changes in future or to correct the errors
  - Always have a comment describing what the function is doing (not how) - helps making it modular
  - If you have to write a long commentary to explain - rethink
- A function can be defined once and used multiple times in the program. This reduces the lines of code that the programmer needs to write.
- The significance of functions becomes clear when the size of program becomes large.

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
- We will consider all functions are defined at the program level (later will discuss functions 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

```
# 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
```

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
- ….

Quiz – Multi Correct

Which all statements are incorrect?

A. The variables used inside function are called local variables.

B. The local variables of a particular function can be used inside other functions, but these cannot be used in global space.

C. The variables used outside function are called super variables.

D. In order to change the value of global variable inside function, keyword global is used.

Quiz – Multi Correct

Which all statements are incorrect?

A. The variables used inside function are called local variables.

B. The local variables of a particular function can be used inside other functions.

C. The variables used outside function are called super variables.

D. In order to change the value of global variable inside function, keyword global is used.

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 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

- 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