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

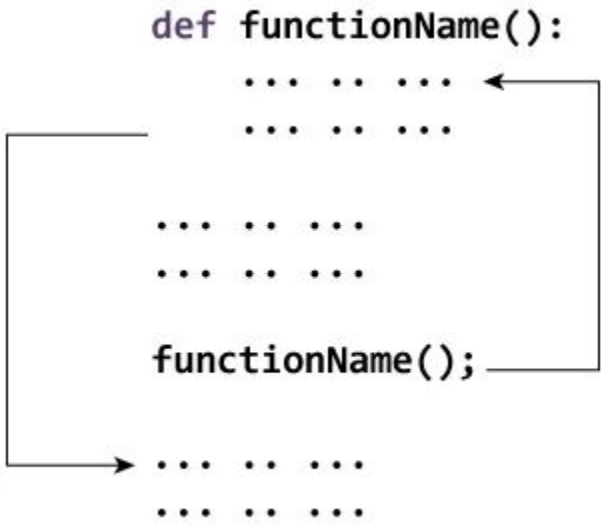
```
fn_name()
```

- 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)



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

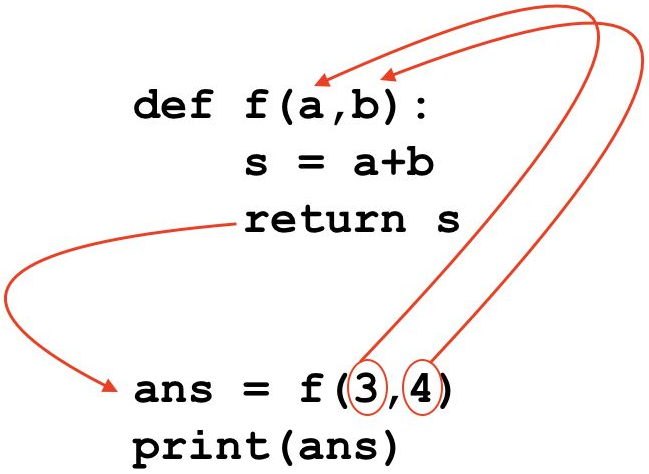
```
print("Red")#1
def f(a, b): #6
    s = a*b #7
    print("Green") #8
    return s #9

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



- 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