FREE EDITION · NOTES + 3 SAMPLE MCQS

CUET · COMPUTER SCIENCE · CLASS XI · CODE 308

Functions

CUET unit: Functions

By **UniDrill** · NCERT-grounded study material

WWW.UNIDRILL.IN

📌 Snapshot

- Functions are the primary tool for achieving modularity and reusability in Python programs.
- User-defined functions differ from Python's built-in functions and the standard library; the function lifecycle covers definition, arguments/parameters, return values, and flow of execution.
- Variable scope is either global or local — a concept NTA directly tests through code-tracing questions.
- Sections on Python's standard library (built-in functions, and the math, random, and statistics modules) are directly examinable as fill-in-the-function or output-prediction items.
- Functions underpin every Python programming task; understanding default parameters, scope rules, and module import syntax is essential for the CUET programming section.

📖 Detailed Notes

2.1 Core concepts

- **Modular programming** is the process of dividing a program into separate, independent blocks of code (sub-problems) with different names and specific functionalities. Program 7-1 (without functions) is contrasted with Program 7-2 (with functions) to illustrate this. (NCERT §7.1, p. 143)
- **Function definition:** A function is a named group of instructions that accomplishes a specific task when invoked. It can be called repeatedly from different places without rewriting the code. (NCERT §7.2, p. 145)
- **Advantages of functions:** (a) Increases readability by organising code into logical blocks; (b) Reduces code length — same code is not repeated; (c) Increases reusability — a function can be called from another function or another program; (d) Enables parallel development by dividing work among team members. (NCERT §7.2.1, p. 146)
- **User-defined functions (UDF):** Functions defined by the programmer to meet specific requirements. Syntax begins with the keyword `def`, followed by function name, optional parameter list in parentheses, and a colon. The body is indented. A function may or may not have parameters; it may or may not return a value. Function

name must be unique and follows identifier naming rules. Function header always ends with a colon (:). (NCERT §7.3, §7.3.1, p. 147)

- **Arguments vs. Parameters:** An argument is a value passed to the function during the function call. A parameter is the variable in the function header that receives the argument. Both refer to the same memory object; they share the same `id()` before any reassignment inside the function. (NCERT §7.3.2, p. 148–149)
- **String as parameters:** Functions can accept string arguments, not just numeric ones. The `+` operator is used to concatenate strings inside a function (Program 7-8). (NCERT §7.3.2(A), p. 151–152)
- **Default parameter:** Python allows assigning a predecided value to a parameter. The default value is used when the function call does not supply the corresponding argument. Default parameters must be trailing parameters — if one parameter has a default value, all parameters to its right must also have default values. A function argument can also be an expression (e.g., `mixedFraction(num+5, deno+5)`), evaluated before the call. (NCERT §7.3.2(B), p. 152–153)
- **Functions returning values:** Functions that do not use `return` are called void functions. The `return` statement returns control to the calling function and optionally returns one or more values. Multiple return values are packed as a tuple. (NCERT §7.3.3, p. 154–156)
- **Flow of execution:** Python executes statements top to bottom. When a function definition is encountered, its body is skipped until the function is called. When a call is encountered, control jumps to the function body, executes it, then returns to the point of the call. A function must be defined before it is called, otherwise a `NameError` is raised. (NCERT §7.3.4, p. 155)
- **Scope of a variable:** The part of the program where a variable is accessible is its scope. (NCERT §7.4, p. 158)
- **Global variable:** Defined outside any function or block; accessible throughout the program. Any modification is permanent and affects all functions. (NCERT §7.4(A), p. 158)
- **Local variable:** Defined inside a function or block; accessible only within that function; exists only while the function executes. A local variable with the same name as a global variable hides the global inside that function. To modify a global variable inside a function, prefix it with the `global` keyword. (NCERT §7.4(B), p. 158–160)
- **Python Standard Library:** A large collection of built-in functions and modules. Functions like `input()`, `print()`, `int()`, `abs()`, `max()`, `min()`, `pow()`, `divmod()`, `sum()`, `len()` are built-in. Additional functionality is available through modules. (NCERT §7.5, §7.5.1, p. 160–162)
- **Modules:** A module is a `.py` file containing a collection of function definitions. Import syntax: `import modulename`. To call a function from a module: `modulename.functionname()`. The `from` statement imports specific functions, allowing direct calls without the module prefix. A module is loaded only once

regardless of how many times it is imported. Key built-in modules: `math`, `random`, `statistics`. (NCERT §7.5.2, p. 162–168)

- **math module:** `math.ceil(x)` — ceiling value; `math.floor(x)` — floor value; `math.fabs(x)` — absolute value (float); `math.factorial(x)` — factorial; `math.gcd(x,y)` — greatest common divisor; `math.pow(x,y)` — x raised to y; `math.sqrt(x)` — square root; `math.sin(x)` — sine in radians. (NCERT §7.5.2, Table 7.2, p. 163–164)
- **random module:** `random.random()` — random float in [0.0, 1.0); `random.randint(x,y)` — random integer between x and y (inclusive); `random.randrange(y)` — random integer between 0 and y; `random.randrange(x,y)` — random integer between x and y. (NCERT §7.5.2, Table 7.3, p. 164–165)
- **statistics module:** `statistics.mean(x)` — arithmetic mean; `statistics.median(x)` — median; `statistics.mode(x)` — mode (most repeated value). (NCERT §7.5.2, Table 7.4, p. 165)
- **Docstring:** A multiline comment (`"""..."""`) added as the first line of a module or function to describe it. Stored in the `__doc__` attribute, accessed via `print(modulename.__doc__)`. (NCERT §7.5.2, p. 167–168)

2.2 Definitions to memorise

Term	Definition	Page
Function	A named group of instructions that accomplishes a specific task when invoked	145
Modular programming	Dividing a program into separate independent blocks of code (sub-problems) with different names and specific functionalities	145
User-defined function	A function defined by the programmer to achieve a specific task as per requirement	147
Argument	A value passed to the function during the function call	148
Parameter	A variable in the function header that receives the argument value	148
Default parameter	A parameter with a predecided value used when no corresponding argument is supplied in the function call	152
Void function	A function that does not return any value (performs task and displays results internally)	154
return statement	Statement that returns control (and optionally value/s) from the function to the calling code	154
Flow of execution	The order in which statements in a program are executed	155
Scope of a variable	The part of the program where a variable is accessible	158
Global variable		158

Term	Definition	Page
	A variable defined outside any function or block; accessible throughout the program	
Local variable	A variable defined inside a function or block; accessible only within that function; ceases to exist after function returns	158
Module	A Python (.py) file containing a collection of function definitions	162
Docstring	A multiline string (`"""..."""`) used to describe a module or function; stored in `__doc__`	167
`from` statement	Used to import only specific function(s) from a module, allowing direct call without module prefix	166
Built-in function	A function pre-defined in Python and available without explicit import (e.g., `print` , `len` , `abs`)	161
`def` keyword	Reserved word that introduces a function definition in Python	147
Function header	The `def name(parameters):` line that introduces a function definition	147
Function body	The indented block of statements that executes when the function is called	147
`global` keyword	Used inside a function to indicate that a name refers to a module-level (global) variable for assignment	159
`math` module	Standard library module providing mathematical functions like `sqrt` , `floor` , `ceil` , `factorial` , `gcd`	163
`random` module	Standard library module providing pseudo-random number generators like `random()` , `randint()` , `randrange()`	164
`statistics` module	Standard library module providing `mean` , `median` , `mode`	165
`__doc__`	Special attribute that stores the docstring of a module or function	167
Function call	An expression that invokes a function with optional arguments	145
Reusability	Property of functions that allows the same code to be used from multiple call sites	146

2.3 Diagrams / processes to remember

- **Figure 7.2 — Calculation of the cost of the tent (p. 144):** Illustrates how a single program is broken into three named blocks (`cyl(h,r)` , `con(l)` , `post_tax_price()`), motivating the concept of modular programming.
- **Figure 7.3 — Argument and parameter pointing to the same value (p. 148):** Shows that `num` (argument) and `n` (parameter) both refer to the same memory object (value 5) before any reassignment inside the function.

- **Figure 7.4 — ID of argument and parameter before and after increment (p. 150):** Before increment, `Number` and `Num` share the same `id`. After `num = num + 5`, `Num` points to a new object (id changes), while `Number` remains unchanged. Demonstrates Python's immutability for integers.
- **Figure 7.5 — Order of execution of statements (p. 156):** Shows numbered execution order for two programs with function calls — the interpreter skips function definitions and executes them only when called.
- **Figure 7.6 — Scope of a variable (p. 158):** Tree diagram showing Variable Scope branching into Global Scope and Local Scope.
- **Figure 7.7 — Types of functions (p. 160):** Tree showing Function branching into Standard Library (Built-in, Module) and User Defined.
- **Table 7.1 — Commonly used built-in functions (p. 161–162):** `abs`, `divmod`, `max`, `min`, `pow`, `sum`, `len` with arguments, return types, and examples. Must be memorised.
- **Tables 7.2, 7.3, 7.4 — math, random, statistics module functions (p. 163–165):** Key functions with syntax, arguments, return values, and example outputs.

2.4 Common confusions / NTA trap points

- **Argument vs. Parameter confusion:** NTA often gives a function call and asks which variable is the "argument" and which is the "parameter." Remember: argument is at the call site; parameter is in the function header. They can have the same name (Program 7-6), which trips students into thinking they are different variables.
- **Default parameter placement rule:** Students confuse which parameters can have defaults. The rule is that default parameters must be trailing — `def f(a, b=1)` is correct but `def f(a=1, b)` is incorrect. NTA presents incorrect headers (like `def calcInterest(principal=1000, rate, time=5)`) as distractors.
- `global` **keyword omission:** A common trap is a function that reads a global variable without issue, but fails when it tries to reassign it without the `global` keyword. Students often assume reading and writing a global variable work the same way inside a function — they do not.
- `import` **vs.** `from ... import`**:** When using `import math`, the call must be `math.sqrt(x)`. When using `from math import sqrt`, the call is simply `sqrt(x)`. NTA tests whether students know that the module prefix is dropped when using `from`.
- **Void function vs. returning** `None`**:** A void function still technically returns `None` in Python. NTA may ask "what does a function that has no `return` statement return?" — the answer is `None`, not "nothing" or "error."
- `math.pow(x,y)` **vs. built-in** `pow(x,y)` **(NCERT Table 7.1 and Table 7.2, p. 162-163).** `math.pow()` always returns a float; the built-in `pow()` returns an integer when both arguments are integers. `pow(x,y,z)` also computes `(x**y) % z`, which `math.pow()` cannot do.

- **A function without `return` returns `None` (NCERT §7.3.3, p. 154).** NTA distractor: "no return → returns 0" — false.
- **`from module import *` is discouraged but legal (NCERT §7.5.2(C), p. 166).** It imports all functions; NCERT mentions star import as an option but recommends specific imports.
- **Multiple return values are packed as a tuple (NCERT §7.3.3, p. 154-156).** A `return a, b` actually returns the tuple `(a, b)`; the caller can unpack with `x, y = func()`.
- **Function must be defined BEFORE the call (NCERT §7.3.4, p. 155).** Forward references raise `NameError`. NTA distractor may suggest "Python supports hoisting" — false.
- **Local variables die when the function returns (NCERT §7.4(B), p. 158-159).** They cannot be accessed after the function ends. NTA traps may show post-function access of a local.

🎯 Practice MCQs

Q1. Which of the following is the correct syntax for defining a user-defined function in Python?

A. `function myFunc(a, b):`

B. `def myFunc(a, b):`

C. `define myFunc(a, b):`

D. `fun myFunc(a, b):`

Q2. Consider the following code: ```python num = 10 def myFunc(): num = 20 print(num) myFunc() print(num) ``` What will be the output?

A. 20 followed by 20

B. 10 followed by 10

C. 20 followed by 10

D. Error, variable num is not defined

Q3. Which of the following statements about default parameters in Python is **correct**?

A. Default parameters must be the leading (first) parameters in the function header.

B. All parameters in a function must have default values.

C. Default parameters must be the trailing parameters in the function header.

D. A function with a default parameter cannot accept any argument during a function call.

📊 PYQ Alignment

Functions is one of the most consistently tested topics in CUET Computer Science, typically contributing 5–7 questions per paper; questions focus on identifying correct function syntax, tracing output through scope rules and function calls, distinguishing argument from parameter, matching built-in/module functions to their outputs, and selecting the correct `import` statement. See the PYQ archive for Computer Science.