Unit 4: Functions

Python Programming (DI02032011)

Milav Dabgar

Lecturer, Dept of ICT
GP Palanpur

April 2, 2026

Unit 4 — Contents

What is a Function?

Definition

A **function** is a named, reusable block of code that performs a specific task. It is defined once and can be called (invoked) multiple times.

Why use functions?

- **Reusability** — write once, use many times
- **Modularity** — break large programs into smaller pieces
- **Readability** — code is easier to understand
- **Maintainability** — fix bugs in one place

Types of functions in Python:

- **Built-in functions** — `print()`, `len()`, `input()`, `max()`, etc.
- **User-defined functions** — defined using `def` keyword
- **Module functions** — from `math`, `random`, `statistics`

Defining and Calling a Function (Su'23 Q.4b, W'24 Q.3b)

Syntax:

```
def function_name(parameters):
    """Docstring (optional)"""
    # function body
    return value    # optional
```

Example:

```
def greet(name):                # define
    print("Hello,", name)

greet("Alice")                  # call  --> Hello, Alice
greet("Bob")                    # call  --> Hello, Bob
```

Function with Return Value (W'24 Q.3b OR)

return statement

return sends a value back to the caller and exits the function.

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

x = add(5, 3)       # x = 8
print(x)

def square(n):
    return n * n

print(square(4))    # 16
print(square(7))    # 49
```

Types of Arguments

Positional Arguments

```
def greet(name, age):
    print(name, "is", age, "years old")

greet("Alice", 20)    # positional
```

Default Arguments

```
def greet(name, msg="Good morning"):
    print(msg + ",", name)

greet("Alice")                  # uses default
greet("Bob", "Good night")   # override default
```

Function: Factorial (W'23 Q.3c, W'24 Q.3c, Su'25 Q.4b)

User-defined function to find factorial

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

num = int(input("Enter number: "))
result = factorial(num)
print("Factorial of", num, "=", result)
```

Example: `factorial(5)` $= 5 \times 4 \times 3 \times 2 \times 1 = 120$

Function: Fibonacci Series (Su'23 Q.3c, W'24 Q.4c OR)

Fibonacci series 0 to N

```
def fibonacci(n):
    a, b = 0, 1
    print("Fibonacci series:")
    while a <= n:
        print(a, end=" ")
        a, b = b, a + b
    print()

n = int(input("Enter N: "))
fibonacci(n)
# For N=20: 0 1 1 2 3 5 8 13
```

Function: Armstrong & Palindrome (Su'23 Q.3c OR, W'23 Q.4c)

Check Armstrong number

```
def is_armstrong(n):
    digits = str(n)
    power = len(digits)
    total = sum(int(d)**power for d in digits)
    return total == n

print(is_armstrong(153))  # True (1^3+5^3+3^3=153)
```

Check palindrome

```
def is_palindrome(n):
    return str(n) == str(n)[::-1]

print(is_palindrome(121))   # True
print(is_palindrome(123))   # False
```

Function: Prime Check (Su'25 Q.4b OR)

Check prime using user-defined function

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

num = int(input("Enter number: "))
if is_prime(num):
    print(num, "is Prime")
else:
    print(num, "is Not Prime")
```

Function: Reverse & Cube of Odds (Su'24 Q.3c OR, W'24 Q.4c)

Reverse the entered value

```
def reverse_value(n):
    return str(n)[::-1]

val = input("Enter value: ")
print("Reversed:", reverse_value(val))
```

Cube of odd numbers 1 to 7

```
def cube_of_odds():
    for i in range(1, 8):
        if i % 2 != 0:
            print(f"Cube of {i} = {i**3}")

cube_of_odds()
```

Local Variables (Su'25 Q.3a)

Local Variable

A variable defined **inside a function** is local. It exists only within that function and cannot be accessed outside.

```
def my_func():
    x = 10          # local variable
    print("Inside:", x)

my_func()
# print(x)    # ERROR: x not defined outside
```

Global Variables (Su'25 Q.3a OR)

Global Variable

A variable defined **outside all functions** is global. Use the `global` keyword inside a function to modify it.

```
count = 0              # global variable

def increment():
    global count       # declare global
    count += 1

increment()
increment()
print("Count:", count)   # Count: 2
```

Scope of Variables — Full Example (W'23 Q.3c OR, W'25 Q.3c)

Local vs Global scope

```
x = 100              # global

def show():
    y = 200          # local
    print("Inside: x =", x, "y =", y)

show()
print("Outside: x =", x)
# print(y)  # ERROR: y not accessible here
```

Variable	Scope	Accessible
x	Global	Everywhere
y	Local (show)	Only inside `show()`

Built-in Functions (W'23 Q.4a OR, W'24 Q.4a)

Common built-in functions

```
# print(), input()
name = input("Name: ")
print("Hello,", name)

# len(), max(), min(), sum()
nums = [3, 7, 1, 9, 4]
print(len(nums))     # 5
print(max(nums))     # 9
print(min(nums))     # 1
print(sum(nums))     # 24

# pow(), abs(), round()
print(pow(2, 8))     # 256
print(abs(-15))      # 15
print(round(3.14159, 2))   # 3.14
```

Math Module (W'23 Q.4a, Su'24 Q.4a, Su'25 Q.3c)

Import and use math module

Use `import math` to access mathematical functions.

```
import math

print(math.sqrt(25))      # 5.0
print(math.pow(2, 10))    # 1024.0
print(math.floor(4.7))    # 4
print(math.ceil(4.2))     # 5
print(math.factorial(5)) # 120
print(math.pi)            # 3.14159...
print(math.log(100, 10)) # 2.0
print(math.sin(math.pi/2)) # 1.0
```

Math Module — Functions Table (W'23 Q.4a, W'25 Q.3c OR)

Function	Description
`math.sqrt(x)`	Square root of x
`math.pow(x, y)`	x raised to power y
`math.floor(x)`	Largest integer $\leq$ x
`math.ceil(x)`	Smallest integer $\geq$ x
`math.factorial(x)`	Factorial of x
`math.log(x, base)`	Logarithm of x to base
`math.sin(x)`	Sine of x (radians)
`math.cos(x)`	Cosine of x (radians)
`math.pi`	Value of $\pi \approx 3.14159$
`math.e`	Value of $e \approx 2.71828$

Random Module (W'24 Q.4a OR, W'25 Q.5b)

Common random module functions

```python
import random

# random float in [0.0, 1.0)
print(random.random())

# random integer: randint(a, b) inclusive
print(random.randint(1, 6))   # dice roll

# random float in [a, b]
print(random.uniform(1.0, 10.0))

# choose random item from list
colors = ["red", "green", "blue"]
print(random.choice(colors))

# shuffle list in place
nums = [1, 2, 3, 4, 5]
random.shuffle(nums)
print(nums)
```

Statistics Module (Su'25 Q.3c OR, W'25 Q.5b OR)

Statistics module functions

```
import statistics as st

data = [4, 8, 15, 16, 23, 42]

print(st.mean(data))       # 18.0 (average)
print(st.median(data))     # 15.5 (middle value)
print(st.mode([1,2,2,3]))# 2 (most frequent)
print(st.stdev(data))      # standard deviation
print(st.variance(data)) # variance
```

Function	Purpose
`mean()`	Arithmetic mean
`median()`	Middle value
`mode()`	Most frequent value
`stdev()`	Standard deviation

Module in Python (W'23 Q.4b, Su'23 Q.5b)

What is a Module?

A **module** is a file containing Python code (functions, variables, classes) that can be imported into other programs.

```python
# Import entire module
import math
print(math.sqrt(16))    # 4.0

# Import specific function
from math import sqrt, pi
print(sqrt(25))   # 5.0
print(pi)         # 3.14159...

# Import with alias
import statistics as st
data = [1, 2, 3, 4, 5]
print(st.mean(data))    # 3.0
```

PYQ Practice — 3-Mark Questions

3-Mark PYQ Bank

1. Explain local variable with example *(Su'25)*
2. Explain global variable with example *(Su'25 OR)*
3. List Python standard library mathematical functions *(W'23, Su'24 OR)*
4. Describe Python math module with Python code example *(Su'24)*
5. Describe built-in functions `max()`, `input()`, `pow()` with example *(W'24)*
6. Explain random module with various functions *(W'24 OR)*
7. Explain built-in functions in Python *(W'23 OR, Su'24 OR)*

PYQ Practice — 4-Mark Questions

4-Mark PYQ Bank

1. Define function. Explain user-defined function with suitable example *(Su'23)*
2. Explain how to define and call a user-defined function *(W'24)*
3. Explain return statement in function handling *(W'24 OR)*
4. User-defined function to find factorial of a number *(Su'25, W'25 OR)*
5. User-defined function to check if number is prime *(Su'25 OR)*
6. User-defined function to print Fibonacci series 0 to N *(W'25)*
7. Explain local and global variables with suitable examples *(Su'23 OR)*
8. Write Python program that explains scope of a variable *(Su'24)*
9. Explain math module and random module with example *(Su'23)*
10. Explain Module in Python with Python code example *(W'23)*
11. Explain Random module in Python *(W'25)*
12. Explain statistics module in Python *(W'25 OR)*

PYQ Practice — 7-Mark Questions

7-Mark PYQ Bank

1. Create user-defined function to print Fibonacci series 0 to N *(Su'23, W'24 OR)*
2. Determine if number is Armstrong or palindrome using function *(Su'23 OR, W'23, Su'24)*
3. Create user-defined function to find factorial *(W'23, W'24)*
4. Explain local and global variables; concept of scope *(W'23 OR ×2, W'25)*
5. Write program using function that reverses the entered value *(Su'24 OR)*
6. Create function that prints cube of all odd numbers 1–7 *(W'24)*
7. List math module functions; develop program demonstrating them *(Su'25)*
8. List statistics module functions; develop program demonstrating them *(Su'25 OR)*
9. List Python standard library mathematical functions; explain any three *(W'25 OR)*

Worked Solution: Factorial (W'23 Q.3c, W'24 Q.3c)

Complete factorial program

```
def factorial(n):
    """Returns factorial of n"""
    if n == 0 or n == 1:
        return 1
    fact = 1
    for i in range(2, n + 1):
        fact *= i
    return fact

# Main program
num = int(input("Enter a positive integer: "))
if num < 0:
    print("Factorial not defined for negative")
else:
    print(f"Factorial of {num} = {factorial(num)}")
```

Worked Solution: Math Module Program (Su'25 Q.3c)

Program demonstrating math module functions

```
import math

n = float(input("Enter a number: "))

print("Square root:", math.sqrt(n))
print("Floor:", math.floor(n))
print("Ceil:", math.ceil(n))
print("log base 10:", math.log10(n))
print("Pi:", math.pi)
print("Power (n^2):", math.pow(n, 2))

# Hypotenuse of right triangle
a = 3; b = 4
print("Hypotenuse:", math.hypot(a, b))   # 5.0
```

Lambda / Anonymous Functions (W'25, Su'25 OR — 4M)

Syntax

`lambda` *arguments* `:` *expression*
Lambda creates a small, unnamed (anonymous) function in a single line.

```
# Regular function
def add(a, b):
    return a + b

# Equivalent lambda
add = lambda a, b: a + b
print(add(3, 5))      # 8

square = lambda x: x ** 2
print(square(4))      # 16

is_even = lambda n: n % 2 == 0
print(is_even(7))     # False
print(is_even(8))     # True
```

Lambda with map() & filter() (W'25 — 4M)

```
nums = [1, 2, 3, 4, 5, 6]

# map(): apply function to all elements
squares = list(map(lambda x: x**2, nums))
print(squares)   # [1, 4, 9, 16, 25, 36]

doubled = list(map(lambda x: x*2, nums))
print(doubled)   # [2, 4, 6, 8, 10, 12]

# filter(): keep elements that match
evens = list(filter(lambda x: x%2==0, nums))
print(evens)     # [2, 4, 6]

positives = list(filter(lambda x: x>3, nums))
print(positives) # [4, 5, 6]
```

*args — Variable Positional Arguments (W'25 OR — 4M)

Concept

*args allows a function to accept **any number** of positional arguments. Collected as a **tuple** inside the function.

```
def total(*nums):
    s = 0
    for n in nums:
        s += n
    return s

print(total(1, 2, 3))           # 6
print(total(10, 20, 30, 40))    # 100
print(total(5))                 # 5

def display(*items):
    for item in items:
        print(item, end=" ")
    print()
```

**kwargs — Variable Keyword Arguments (W'25 OR — 4M)

Concept

kwargs accepts any number of **keyword arguments. Collected as a **dictionary** inside the function.

```
def student_info(**details):
    for key, value in details.items():
        print(f"{key}: {value}")

student_info(name="Raj", age=20, gpa=8.5)
# name: Raj
# age: 20
# gpa: 8.5

# Combining *args and **kwargs
def mixed(*args, **kwargs):
    print("Args:", args)
    print("Kwargs:", kwargs)

mixed(1, 2, 3, x=10, y=20)
```

Recursion — Concept (W'24, Su'25 — 4M)

What is Recursion?

A function that **calls itself** is called a recursive function. Every recursive function must have a **base case** to stop recursion.

Structure

```
def recursive_fn(n):
if base_case:
return value
return expression with recursive_fn(n-1)
```

Key Terms

Base Case: Condition to stop recursion (prevents infinite loop)
Recursive Case: The function calls itself with simpler input
Call Stack: Each call is pushed; returns pop in reverse order

Recursive Function: Factorial (W'24, Su'25 OR — 7M)

```python
def factorial(n):
    # Base case
    if n == 0 or n == 1:
        return 1
    # Recursive case
    return n * factorial(n - 1)

print(factorial(5))    # 120
print(factorial(0))    # 1
print(factorial(6))    # 720
```

Call Trace: factorial(4)

`factorial(4)` → `4 * factorial(3)`
`factorial(3)` → `3 * factorial(2)`
`factorial(2)` → `2 * factorial(1)`
`factorial(1)` → 1 (base case)
Back: $2 \times 1 = 2 \rightarrow 3 \times 2 = 6 \rightarrow 4 \times 6 = \mathbf{24}$

Recursive Function: Fibonacci (Su'24, W'25 OR — 7M)

```python
def fibonacci(n):
    if n == 0:
        return 0      # base case 1
    if n == 1:
        return 1      # base case 2
    return fibonacci(n-1) + fibonacci(n-2)

# Print first 10 terms
for i in range(10):
    print(fibonacci(i), end=" ")
# 0 1 1 2 3 5 8 13 21 34
```

Note

Recursive Fibonacci is elegant but **slow** for large n (repeated calls). Iterative version is more efficient for large inputs.

Recursion vs Iteration (W'24 OR, Su'25 — 4M)

Comparison Table

Feature	Recursion	Iteration
Approach	Function calls itself	Loops (`for`/`while`)
Base case	Mandatory (no infinite calls)	Loop condition
Memory	Uses call stack (more)	Less memory
Speed	Usually slower	Usually faster
Code size	Often shorter	Can be longer
Debugging	Harder to trace	Easier
Use case	Tree, factorial, Fibonacci	Counting, sum, search

Recursive: Sum of Digits & Power (Su'25 OR — 4M)

```python
# Recursive sum of digits
def sum_digits(n):
    if n == 0:
        return 0
    return (n % 10) + sum_digits(n // 10)

print(sum_digits(1234))    # 10
print(sum_digits(999))     # 27

# Recursive power: base^exp
def power(base, exp):
    if exp == 0:
        return 1
    return base * power(base, exp - 1)

print(power(2, 8))     # 256
print(power(3, 4))     # 81
```

String Module (W'24 OR, Su'25 — 3M)

```
import string

print(string.ascii_letters)
# abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

print(string.ascii_lowercase)  # a-z
print(string.ascii_uppercase)  # A-Z
print(string.digits)           # 0123456789
print(string.punctuation)      # !"#$%&'()*+,-./:;<=>?

# Practical use: password validation
def is_strong(pwd):
    return (any(c in string.ascii_uppercase for c in pwd)
        and any(c in string.digits for c in pwd)
        and len(pwd) >= 8)

print(is_strong("Hello123"))   # True
print(is_strong("hello"))      # False
```

os Module — File System Basics (W'24 — 3M)

```python
import os

# Current working directory
print(os.getcwd())          # /home/user/python

# Change directory
os.chdir("/tmp")

# List files and folders
print(os.listdir("."))      # ['file1.py', 'data/']

# Create and remove directory
os.mkdir("new_folder")
os.rmdir("new_folder")

# Check if path exists
print(os.path.exists("data.txt"))  # True/False

# Get file size
print(os.path.getsize("data.txt")) # bytes

# Join paths safely
path = os.path.join("folder", "file.txt")
```

datetime Module (Su'25 OR — 3M)

```python
from datetime import datetime, date, timedelta

# Current date and time
now = datetime.now()
print(now)
# 2025-06-15 10:30:45.123456

print(now.year, now.month, now.day)
print(now.hour, now.minute, now.second)

# Today's date
today = date.today()
print(today)                    # 2025-06-15

# Date arithmetic
tomorrow = today + timedelta(days=1)
deadline = today + timedelta(weeks=2)
print(tomorrow, deadline)

# Format date as string
print(now.strftime("%d/%m/%Y"))   # 15/06/2025
```

Function: GCD & LCM (Su'24, W'25 — 4M)

```python
# GCD using Euclidean algorithm
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

# LCM using GCD
def lcm(a, b):
    return (a * b) // gcd(a, b)

x = int(input("First number: "))
y = int(input("Second number: "))
print(f"GCD of {x} and {y} = {gcd(x, y)}")
print(f"LCM of {x} and {y} = {lcm(x, y)}")
# For x=12, y=8:
# GCD = 4,  LCM = 24
```

Function: Perfect Number & Abundant (W'25, Su'25 — 4M)

```python
def is_perfect(n):
    s = sum(i for i in range(1, n) if n % i == 0)
    return s == n

# Check perfect numbers up to 1000
print("Perfect numbers up to 1000:")
for num in range(1, 1001):
    if is_perfect(num):
        print(num, end=" ")
# 6 28 496

# Explanation:
# 6 = 1+2+3, 28 = 1+2+4+7+14
def divisor_sum(n):
    return sum(i for i in range(1, n) if n%i==0)

print(divisor_sum(6))     # 6  (perfect)
print(divisor_sum(12))    # 16 (abundant: sum > n)
```

Function: Digit Operations (W'25 OR — 4M)

```
def digit_sum(n):
    """Sum of all digits of n"""
    total = 0
    while n > 0:
        total += n % 10
        n //= 10
    return total

def digit_count(n):
    """Count digits in n"""
    count = 0
    while n != 0:
        n //= 10
        count += 1
    return count

def digit_product(n):
    """Product of digits"""
    prod = 1
    while n > 0:
        prod *= n % 10
        n //= 10
    return prod
```

Function: Leap Year & Temperature Conversion (Su'25 — 4M)

```python
def is_leap_year(year):
    return (year%4==0 and year%100!=0) or (year%400==0)

def celsius_to_fahrenheit(c):
    return (c * 9/5) + 32

def fahrenheit_to_celsius(f):
    return (f - 32) * 5/9

def celsius_to_kelvin(c):
    return c + 273.15

y = int(input("Year: "))
if is_leap_year(y):
    print(f"{y} is a Leap Year")
else:
    print(f"{y} is NOT a Leap Year")

c = float(input("Celsius: "))
print(f"Fahrenheit: {celsius_to_fahrenheit(c):.2f}")
print(f"Kelvin:     {celsius_to_kelvin(c):.2f}")
```

Creating & Using Custom Modules (W'23, Su'23 — 4M)

```python
# File: mymath.py (our custom module)
def add(a, b):    return a + b
def sub(a, b):    return a - b
def mul(a, b):    return a * b
def div(a, b):    return a / b if b != 0 else "Error"
PI = 3.14159

# In another file: main.py
import mymath
print(mymath.add(5, 3))          # 8
print(mymath.PI)                  # 3.14159

# Selective import
from mymath import mul, div
print(mul(4, 7))                  # 28

# Import with alias
import mymath as mm
print(mm.sub(10, 4))              # 6
```

Worked: Lambda, map, filter Program (W'25 — 4M)

```
# Problem: Given a list of marks, find
# (a) squares of all marks
# (b) marks above 40 (pass)
# (c) double the failing marks

marks = [35, 72, 48, 19, 85, 40, 61]

squares = list(map(lambda x: x**2, marks))
print("Squares:", squares)

passed = list(filter(lambda x: x >= 40, marks))
print("Passed:", passed)

failed = list(filter(lambda x: x < 40, marks))
doubled = list(map(lambda x: x*2, failed))
print("Failed doubled:", doubled)

# Sorted using lambda key
marks.sort(key=lambda x: -x)  # descending
print("Sorted:", marks)
```

Worked: Recursive GCD (Su'24, W'25 — 4M)

```
def gcd(a, b):
    if b == 0:          # base case
        return a
    return gcd(b, a % b)    # recursive case

def lcm(a, b):
    return (a * b) // gcd(a, b)

# Test
print(gcd(48, 18))      # 6
print(gcd(100, 75))     # 25
print(lcm(4, 6))        # 12
print(lcm(12, 18))      # 36
```

Call Trace: gcd(48, 18)

`gcd(48,18)` → `gcd(18,12)` → `gcd(12,6)` → `gcd(6,0)` → **6**

Worked: Armstrong Number using Function (Su'23, W'23 — 7M)

```python
def is_armstrong(n):
    digits = str(n)
    power = len(digits)
    total = sum(int(d)**power for d in digits)
    return total == n

# Print all Armstrong numbers up to 999
print("Armstrong numbers up to 999:")
for i in range(1, 1000):
    if is_armstrong(i):
        print(i, end=" ")
# 1 2 3 4 5 6 7 8 9 153 370 371 407

# Single check
n = int(input("Enter number: "))
if is_armstrong(n):
    print(f"{n} is an Armstrong number")
else:
    print(f"{n} is NOT an Armstrong number")
```

Worked: Default & Keyword Arguments (W'24, Su'25 — 4M)

```python
# Default arguments
def greet(name, msg="Hello"):
    print(f"{msg}, {name}!")

greet("Raj")                # Hello, Raj!
greet("Priya", "Hi")        # Hi, Priya!
greet("Sam", msg="Hey")     # Hey, Sam!

# Keyword arguments (named)
def rectangle(length, width, unit="cm"):
    area = length * width
    perimeter = 2 * (length + width)
    print(f"Area={area} {unit}^2")
    print(f"Perimeter={perimeter} {unit}")

rectangle(5, 3)
rectangle(width=4, length=7, unit="m")
```

Worked: Statistics Module Program (Su'25 OR, W'25 — 4M)

```python
import statistics

marks = [85, 72, 91, 68, 78, 85, 90, 65, 72, 80]

print(f"Mean:   {statistics.mean(marks):.2f}")
print(f"Median: {statistics.median(marks):.2f}")
print(f"Mode:   {statistics.mode(marks)}")
print(f"Stdev:  {statistics.stdev(marks):.2f}")
print(f"Variance: {statistics.variance(marks):.2f}")

# Manual mean (without library)
n = len(marks)
manual_mean = sum(marks) / n
print(f"Manual Mean: {manual_mean:.2f}")

# Count above average
above = [m for m in marks if m > manual_mean]
print(f"Above average: {len(above)} students")
```

Worked: Math Module Programs (W'23, Su'24, W'25 — 4M)

```python
import math

# Quadratic equation roots: ax^2 + bx + c = 0
a, b, c = map(float, input("a b c: ").split())
discriminant = b**2 - 4*a*c
if discriminant > 0:
    r1 = (-b + math.sqrt(discriminant)) / (2*a)
    r2 = (-b - math.sqrt(discriminant)) / (2*a)
    print(f"Roots: {r1:.2f}, {r2:.2f}")
elif discriminant == 0:
    r = -b / (2*a)
    print(f"Equal roots: {r:.2f}")
else:
    print("Complex roots (no real solution)")

# Useful math functions
print(math.gcd(48, 36))       # 12
print(math.factorial(6))      # 720
print(math.comb(5, 2))        # 10   (5C2)
```

Worked: Random Module — Guessing Game (W'24 OR, W'25 — 4M)

```
import random

# Number guessing game
secret = random.randint(1, 100)
attempts = 0
print("Guess the number (1-100)!")

while True:
    guess = int(input("Your guess: "))
    attempts += 1
    if guess < secret:
        print("Too low!")
    elif guess > secret:
        print("Too high!")
    else:
        print(f"Correct! ({attempts} attempts)")
        break

# Other random uses
colors = ["red", "blue", "green", "yellow"]
print(random.choice(colors))    # random color
nums = list(range(1, 11))
```

Worked: Scope — LEGB Rule (W'25, Su'25 — 4M)

LEGB: Scope Resolution Order

Local → **E**nclosing → **G**lobal → **B**uilt-in

```
count = 0           # global variable

def increment():
    global count    # access global
    count += 1

def outer():
    x = 10          # enclosing scope
    def inner():
        nonlocal x   # modify enclosing
        x += 5
        print("inner x:", x)
    inner()
    print("outer x:", x)

increment(); increment()
print("count:", count)     # 2
outer()     # inner x: 15, outer x: 15
```

Quick Reference: Python Standard Modules

Module	Import	Key Functions
math	`import math`	`sqrt, pow, floor, ceil, pi, factorial, gcd, comb`
random	`import random`	`random(), randint(), choice(), shuffle(), seed()`
statistics	`import statistics`	`mean, median, mode, stdev, variance`
string	`import string`	`ascii_letters, digits, punctuation`
os	`import os`	`getcwd, listdir, mkdir, path.exists`
datetime	`from datetime import datetime, date`	`now(), today(), strftime()`

Import Styles

`import math` | `from math import sqrt, pi` | `import math as m`

Quick Reference: Function Types in Python

Type	Description & Example
Built-in	Pre-defined: `len()`, `print()`, `input()`, `type()`, `range()`
User-defined	Defined with `def`: `def greet(): ...`
Lambda	Anonymous: `square = lambda x: x**2`
Recursive	Calls itself: `def fact(n): return n*fact(n-1)`
Higher-order	Takes/returns functions: `map()`, `filter()`
Module functions	From libraries: `math.sqrt()`, `random.randint()`

Argument Types

Positional: `add(3, 5)` **Keyword:** `greet(name="Raj")` **Default:** `def f(x=10)`
***args:** tuple **kwargs:** dict

Common Mistakes in Functions (Exam Guide)

Mistakes

× No base case in recursion
× Modifying global without `global`
× Using variable before definition
× Returning inside loop prematurely
× Wrong number of arguments
× Calling function before defining

Correct Patterns

✓ Always have `if n==0: return 1`
✓ Use `global x` to modify global
✓ Define before calling
✓ Use `return` at end of function
✓ Match parameter count
✓ Put function defs at top of file

Unit 4 — Rapid Revision Flash Sheet

Concept	Key Point
Function definition	`def name(params): body`
Return value	`return expr` (or `None` if absent)
Default argument	`def f(x=10):` — must come last
Lambda	`lambda x: x**2` — single expression
*args	Variable positional → tuple inside
**kwargs	Variable keyword → dict inside
Recursion	Function calls itself; needs base case
Local scope	Variable inside function only
Global keyword	`global x` to modify global inside fn
math module	`sqrt, pow, floor, ceil, pi, factorial`
random module	`randint, choice, shuffle, random()`
statistics module	`mean, median, mode, stdev`

GTU Weightage (Unit 4 — 16%)

3-mark: definitions, modules 4-mark: programs 7-mark: full programs

Worked: Recursive Palindrome Check (W'23 OR, Su'24 — 4M)

```python
def is_palindrome(s):
    """Check if string is palindrome recursively"""
    s = s.lower()
    if len(s) <= 1:
        return True          # base case
    if s[0] != s[-1]:
        return False         # mismatch
    return is_palindrome(s[1:-1])  # recurse on middle

# Test
words = ["madam", "racecar", "hello", "level"]
for w in words:
    result = "Palindrome" if is_palindrome(w) else "Not"
    print(f"{w}: {result}")
# madam: Palindrome
# racecar: Palindrome
# hello: Not
# level: Palindrome
```

Worked: Prime Numbers — All Methods (Su'25, W'24 OR — 7M)

```python
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5)+1):
        if n % i == 0:
            return False
    return True

# Check single number
n = int(input("Number: "))
print(f"{n} is {'Prime' if is_prime(n) else 'Not Prime'}")

# Print all primes up to N
n = int(input("Up to: "))
primes = [i for i in range(2, n+1) if is_prime(i)]
print("Primes:", primes)
print("Count:", len(primes))

# Twin primes (differ by 2)
twins = [(i,i+2) for i in range(2,n) if is_prime(i) and is_prime
    (i+2)]
print("Twin primes:", twins[:5])
```

Worked: Number Theory Programs (Su'25 OR — 7M)

```
def is_perfect(n):
    return n > 1 and sum(i for i in range(1,n) if n%i==0) == n

def is_abundant(n):
    return sum(i for i in range(1,n) if n%i==0) > n

def is_armstrong(n):
    d = str(n); p = len(d)
    return sum(int(c)**p for c in d) == n

def is_palindrome_num(n):
    return str(n) == str(n)[::-1]

# Display analysis for 1-500
for n in range(1, 31):
    tags = []
    if is_prime(n):     tags.append("Prime")
    if is_perfect(n):   tags.append("Perfect")
    if is_armstrong(n):tags.append("Armstrong")
    if tags:
        print(f"{n:3d}: {', '.join(tags)}")
```

Worked: Student Report Card Program (Su'25 — 7M)

```
def calculate_grade(marks):
    avg = sum(marks) / len(marks)
    if avg >= 90: return "A+"
    elif avg >= 75: return "A"
    elif avg >= 60: return "B"
    elif avg >= 40: return "C"
    else: return "F (Fail)"

def display_report(name, marks):
    print(f"\n--- Report Card: {name} ---")
    subjects = ["Maths","Science","English","Python","GTU"]
    for sub, m in zip(subjects, marks):
        print(f"{sub:10s}: {m:3d}", end="")
        print(" PASS" if m >= 40 else " FAIL")
    avg = sum(marks)/len(marks)
    print(f"Average: {avg:.1f} | Grade: {calculate_grade(marks)}")

name = input("Student name: ")
marks = [int(input(f"Subject {i+1}: ")) for i in range(5)]
display_report(name, marks)
```

Worked: PYQ 7M — Fibonacci (Recursive + Iterative) (W'25 — 7M)

```python
# Recursive Fibonacci
def fib_recursive(n):
    if n <= 1:
        return n
    return fib_recursive(n-1) + fib_recursive(n-2)

# Iterative Fibonacci
def fib_iterative(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a

n = int(input("Terms: "))
print("Recursive:", [fib_recursive(i) for i in range(n)])
print("Iterative:", [fib_iterative(i) for i in range(n)])
# Both give: [0, 1, 1, 2, 3, 5, 8, 13, ...]
```

Examiner Tip

For 7M: Show both recursive + iterative versions with output. Mention base case explicitly.

Unit 4 — Summary

User-defined

def keyword
Parameters / Arguments
return statement
Default arguments

Scope

Local — inside function
Global — outside all functions
global keyword to modify

Modules

math — sqrt, pow, floor, ceil, pi
random — random, randint, choice
statistics — mean, median, mode

Key Programs

Factorial, Fibonacci
Prime, Armstrong
Palindrome, Reverse
Cube of odds