Noteskarts

Smart Notes. Bright Future.

COMPUTER APPLICATIONS IN PHARMACY

Unit 2 — Control Structures & Functions

Short Notes | Diagrams | Tables | Quick Concepts

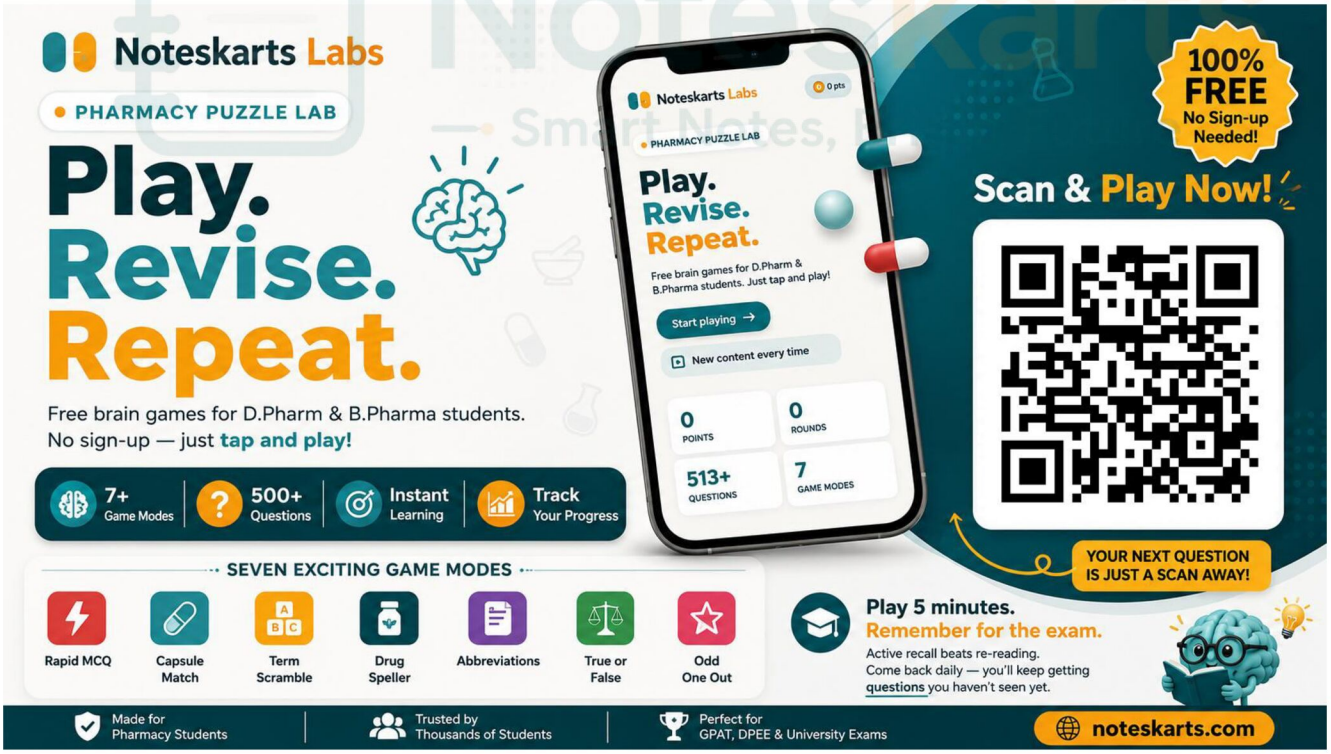
For D.Pharm & B.Pharm Students

www.noteskarts.com

Get Noteskarts Prime — Full Access

Scan the QR code above with your phone camera to download Noteskarts Prime and unlock complete detailed notes, PYQ papers, MCQ tests & video lectures for D.Pharm, B.Pharm, GPAT & Pharmacy Competitive Exams.

Free Bonus — Noteskarts Puzzle Lab



Revise this exact topic through fun MCQ games — Rapid MCQ, Term Scramble, True or False & more. 100% free, no sign-up needed. Scan and play!

Conditional Statements

Conditional statements let a program make decisions and execute different blocks of code depending on whether a condition is True or False. Python uses if, if-else, and if-elif-else for this.

if Statement

Runs a block of code only when the condition is True.

```
dose = 600
if dose > 500:
    print("High dose - check with pharmacist")
```

if-else Statement

Provides an alternate block of code when the condition is False.

```
temperature = 39.5
if temperature > 38:
    print("Fever detected")
else:
    print("Temperature normal")
```

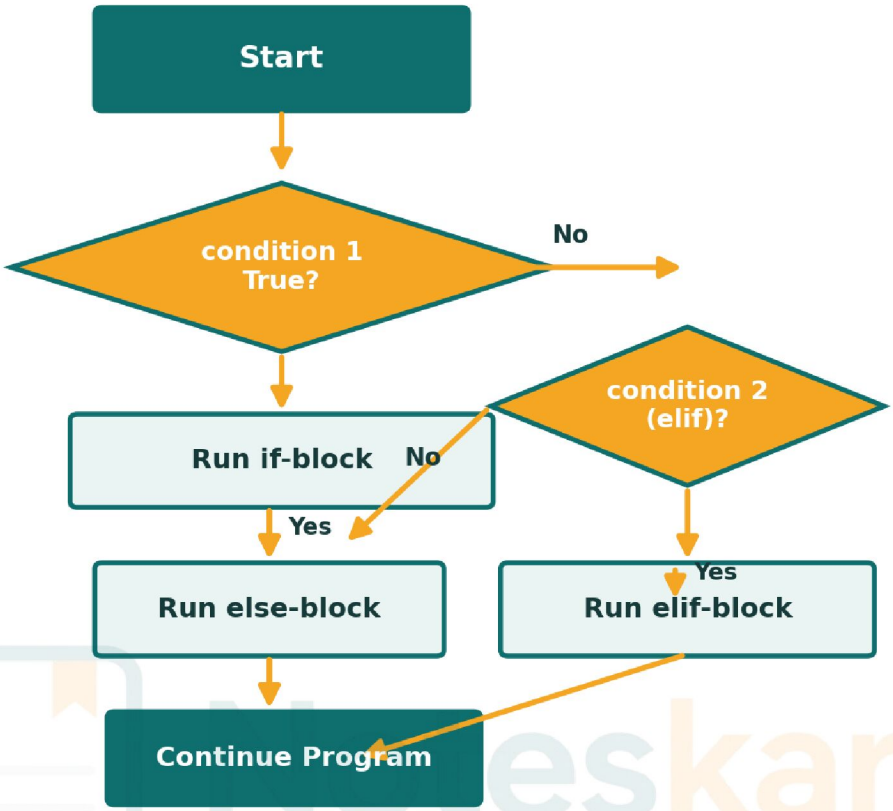
if-elif-else Statement

Used when there are multiple conditions to check in sequence — Python tests each elif only if the previous conditions were False.

```
bp = 145
if bp < 90:
    print("Low BP")
elif bp <= 120:
    print("Normal BP")
elif bp <= 139:
    print("Elevated BP")
else:
    print("High BP - consult doctor")
```

Nested Conditions

A nested condition is an if statement placed inside another if (or elif/else) block. It is used when a decision depends on more than one level of checking.



Worked Example — Drug Dispensing Check

```
age = 65
kidney_ok = True

if age >= 60:
    if kidney_ok:
        print("Reduced dose can be given")
    else:
        print("Consult physician before dosing")
else:
    print("Standard adult dose applies")
```

□ Indentation Matters!

Python uses indentation (spaces), not curly braces { }, to define blocks. Incorrect indentation causes an IndentationError — always keep 4 spaces per nesting level.

Loops - for Loop

A loop repeats a block of code multiple times. A for loop is used when the number of iterations is known or comes from a sequence (list, string, or range()).

Syntax & range()

```
for i in
range(1, 6):
    print(i)
# Output: 1
2 3 4 5
```

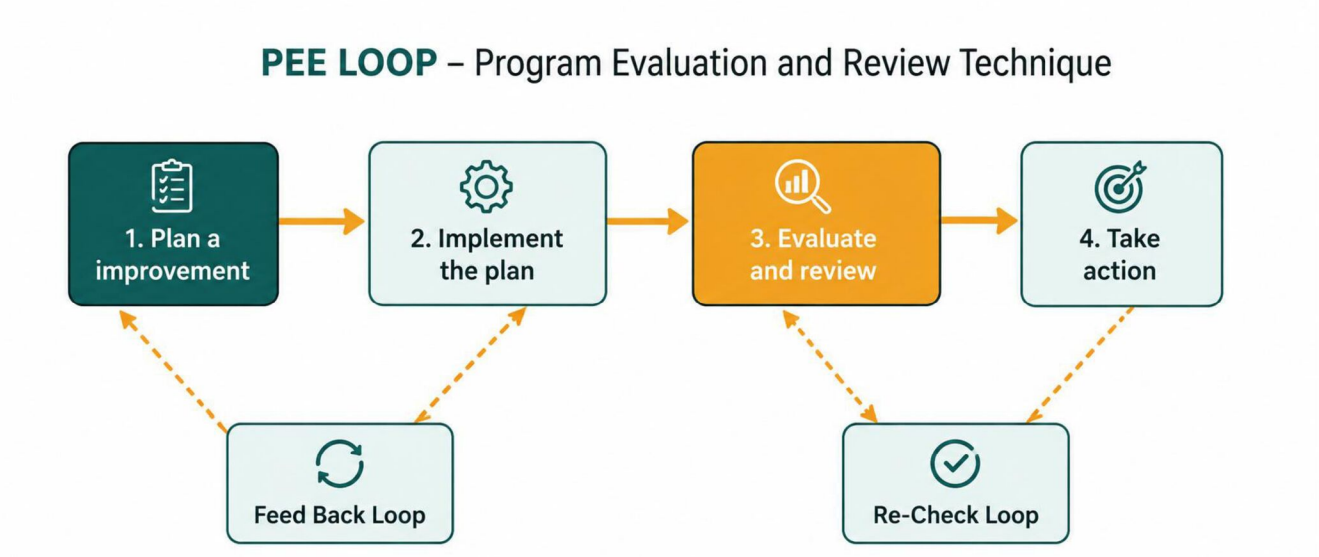
range() form	Meaning
`range(5)`	0,1,2,3,4 (starts at 0)
`range(2,6)`	2,3,4,5
`range(0,10,2)`	0,2,4,6,8 (step=2)

Looping Over a List

```
drugs = ["Paracetamol", "Ibuprofen", "Aspirin"]
for d in drugs:
    print("Tablet:", d)
```


Loops - while Loop

A while loop repeats a block of code as long as a condition remains True. Used when the number of iterations is NOT known in advance.



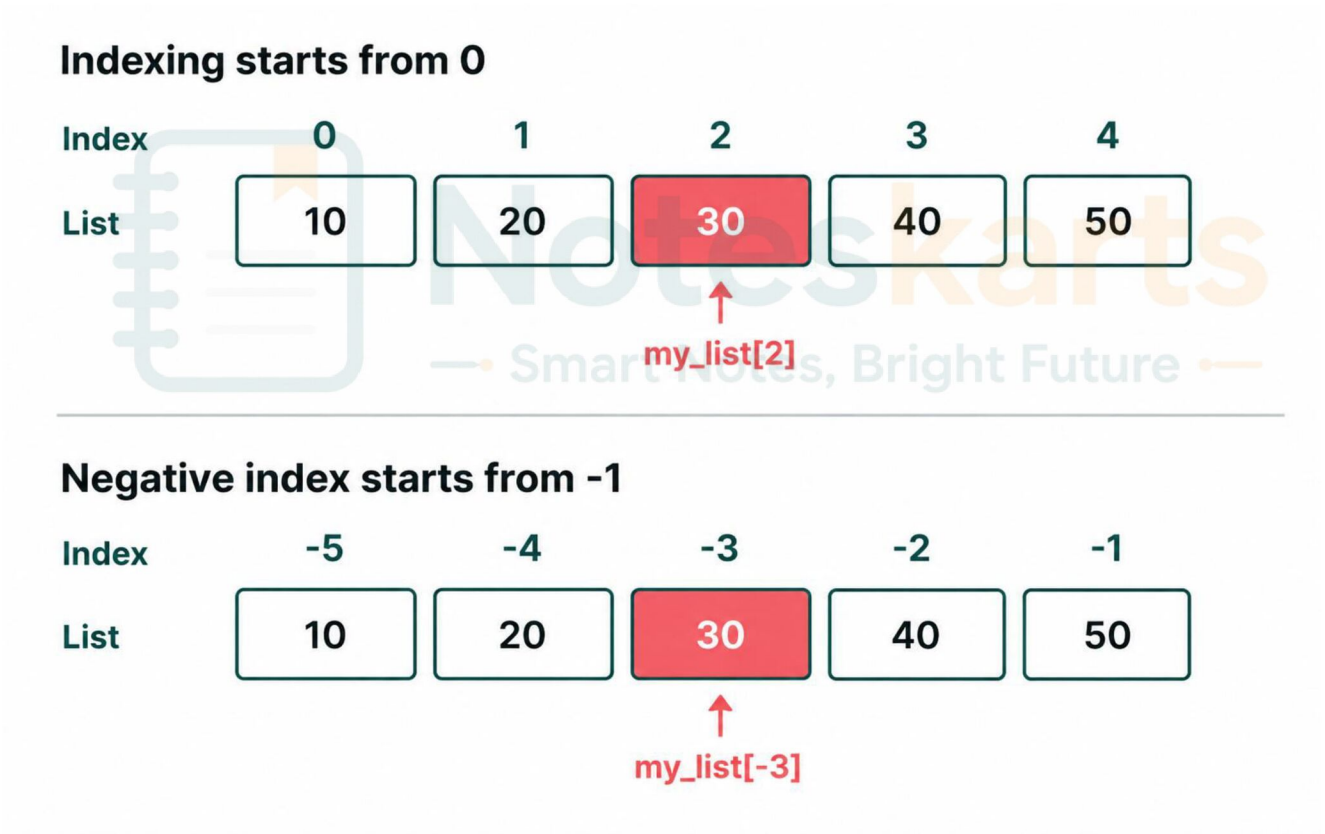
```
stock = 5
while stock > 0:
    print("Dispensing 1 strip, remaining:", stock)
    stock = stock - 1
print("Out of stock!")
```

□ Common Mistake

Forgetting to update the loop variable (e.g., stock = stock - 1) inside a while loop causes an INFINITE LOOP that never stops.

Break and Continue Statements

Statement	Effect
`break`	Stops (exits) the loop completely, immediately
`continue`	Skips the current iteration only, loop continues



break Example

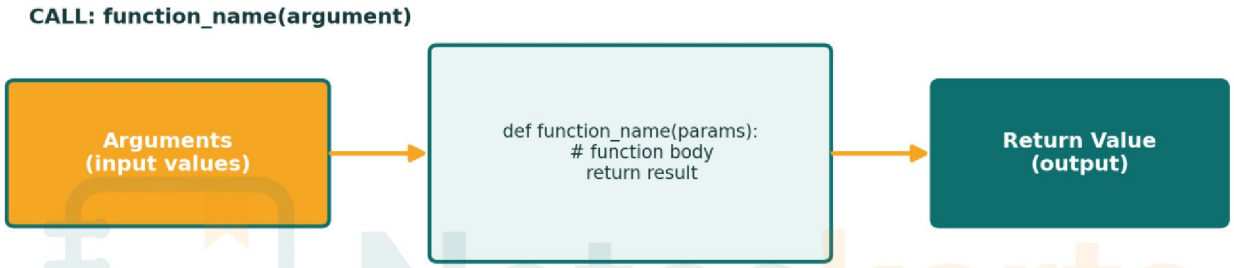
```
batches = ["B1", "B2", "EXPIRED", "B4"]
for b in batches:
    if b == "EXPIRED":
        break
    print("Checking batch:", b)
# Output: Checking batch: B1, Checking batch: B2
```

continue Example

```
batches = ["B1", "B2", "EXPIRED", "B4"]
for b in batches:
    if b == "EXPIRED":
        continue
    print("Checking batch:", b)
# Output: B1, B2, B4 (EXPIRED skipped, loop continues)
```

Defining and Calling Functions

A function is a reusable, named block of code that performs a specific task. Functions avoid repeating the same code again and again, and make programs modular and easy to maintain.



Defining a Function

```
def greet_patient():
    print("Welcome to the pharmacy!")
```

Calling a Function

```
greet_patient()   # calling / invoking the function
# Output: Welcome to the pharmacy!
```

□ Why Use Functions?

Reusability (write once, use many times), better organisation of large programs into smaller logical pieces, and easier debugging & testing of each piece separately.

Passing Arguments & Returning Values

Parameters vs Arguments

A parameter is the variable listed inside the function definition. An argument is the actual value passed to the function when it is called.

```
def calculate_dose(weight, dose_per_kg):   # weight, dose_per_kg =
parameters
    total = weight * dose_per_kg
    return total

result = calculate_dose(60, 5)             # 60, 5 = arguments
print("Total dose:", result, "mg")
# Output: Total dose: 300 mg
```

The return Statement

return sends a value back from the function to the place it was called from, so the result can be stored or reused. A function without return automatically returns None.

Aspect	print()	return
Purpose	Only displays output on screen	Sends value back for further use
Reusable?	No — value is lost after display	Yes — value can be stored in a variable

Types of Function Arguments

Type	Description	Example
Positional	Passed in order they are defined	calc_dose(60, 5)
Default	Has a preset value if not supplied	def f(dose=500):
Keyword	Passed by naming the parameter	calc_dose(weight=60, dose_per_kg=5)
Variable-length (*args)	Accepts any number of extra values	def f(*meds):

Type	Description	Example

```
def calc_dose(weight,
dose_per_kg=5):   # default
argument
    return weight *
dose_per_kg

print(calc_dose(50))
# uses default 5 -> 250
print(calc_dose(50,
dose_per_kg=8))  # keyword
argument -> 400
```

Modular Program - Dosage Calculation

A modular program breaks a task into small functions that each do one job, then calls them together. Below is a simple weight-based dosage calculator built using functions, conditions and a loop.

```
def get_dose_per_kg(age):
    if age < 12:
        return 5          # children - lower dose/kg
    elif age < 60:
        return 10         # adults
    else:
        return 7          # elderly - reduced dose

def calculate_dose(weight, dose_per_kg):
    return weight * dose_per_kg

def dosage_report(name, age, weight):
    rate = get_dose_per_kg(age)
    dose = calculate_dose(weight, rate)
    print(name, "-> Age:", age, "| Dose:", dose, "mg")
    return dose

patients = [("Aarav", 8, 22), ("Meera", 35, 60), ("Ramesh", 68, 55)]
for p_name, p_age, p_weight in patients:
    dosage_report(p_name, p_age, p_weight)
```

Sample Output:

```
Aarav -> Age: 8 | Dose: 110 mg
Meera -> Age: 35 | Dose: 600 mg
Ramesh -> Age: 68 | Dose: 385 mg
```

□ Concepts Used

if-elif-else (age-based rate), functions with parameters & return values, a for loop over a list of tuples, and function calls within another function — a full revision of this unit.

Modular Program - BMI Calculation

Body Mass Index (BMI) = weight (kg) / height (m)^2. This modular program calculates BMI and classifies it using if-elif-else, wrapped inside reusable functions.

```
def calculate_bmi(weight, height):
    return weight / (height ** 2)

def classify_bmi(bmi):
    if bmi < 18.5:
        return "Underweight"
    elif bmi < 25:
        return "Normal"
    elif bmi < 30:
        return "Overweight"
    else:
        return "Obese"

def bmi_report(name, weight, height):
    bmi = calculate_bmi(weight, height)
    category = classify_bmi(bmi)
    print(name, "- BMI:", round(bmi, 1), "->", category)

bmi_report("Priya", 58, 1.60)
bmi_report("Karan", 90, 1.75)
```

Sample Output:

```
Priya - BMI: 22.7 -> Normal
Karan - BMI: 29.4 -> Overweight
```

□ Why Modular?

calculate_bmi() and classify_bmi() can be reused independently in other programs (e.g., a diet-advice tool) without rewriting the logic — that's the power of functions.

Important Points for Examination

- if / if-else / if-elif-else let a program branch based on conditions; elif is checked only if earlier conditions were False.
- Python uses INDENTATION (not braces) to define code blocks — incorrect indentation causes an IndentationError.
- Nested conditions are an if inside another if/else, used for multi-level decisions.

- for loop -> fixed number of iterations over a sequence (list, string, range()).
- while loop -> repeats until a condition becomes False; must update the loop variable to avoid an infinite loop.
- break -> exits the loop completely; continue -> skips only the current iteration.
- A function is defined using def and is reused via a function call; parameters (in definition) receive arguments (at call time).
- return sends a value back for reuse; print() only displays it and the value is lost.
- Argument types: positional, default, keyword, and *args (variable-length).
- Modular programs split a task into small functions — e.g., calculate_dose() and classify_bmi() — for reusability and easier debugging.
- BMI = weight(kg) / height(m) squared; classified as Underweight / Normal / Overweight / Obese.