Control statements:

- **for loop:**

```
     for loop:=>for loop is not a continous loop.(there is a limit)
#for loop is used to iterate over a sequence (like a list, tuple, string, or range).
#syntax=>
         # for <var> in range(sp,en):
         # logic here
```

- while loop:

Repeats a block of code while a condition is True.

Feature	for Loop	while Loop
Use Case	Used when you **know in advance** how many times to iterate.	Used when you **don't know in advance** how many times to loop—based on a condition.
Syntax	`for item in iterable:`	`while condition:`
Example	`for i in range(5):`	`while i < 5:`
Termination	Automatically stops after iterating over the sequence.	Stops when the condition becomes `False`.
Common Usage	Iterating over strings, lists, tuples, dictionaries, etc.	Repeating a block until a condition is no longer true.

Performance	More concise and efficient for iterables.	Can be less efficient if condition is not well controlled.
Risk of Infinite Loop	Rare, since it loops over a fixed sequence.	More prone to infinite loops if condition never becomes `False`.
Control Statements	Supports `break`, `continue`, and `else`.	Also supports `break`, `continue`, and `else`.

Conditional statements:

- If : Used to test a condition. If the condition is **True**, the block runs.
- Else: Used when **all previous conditions are False**. Acts as a default/fallback.
- Elif : Used to test **multiple conditions** after the initial `if`. Only the **first** `True` **condition** executes.
- Break: The `break` statement is used to **exit or terminate a loop immediately**, even if the loop condition is still `True`

Keyword	Purpose	Behavior	Common Use Case
`break`	Exits the nearest enclosing loop immediately	Terminates the loop even if the condition is still `True`	Stop the loop when a condition is met
`continue`	Skips the rest of the current iteration and moves to the next one	Skips the remaining code in the loop for current iteration	Skip specific cases inside a loop

`pass`	Placeholder for future code	Does nothing; just a syntactic placeholder	Used when code is syntactically required but not written yet

Use of Functions

- `function are used for -->reusability`

In Python there are 2 types of Functions:

- `1. Built_in_functions -> given by the Python community`
- `2. User_defined_functions -> we are going to create the functions -> and In User defined functions there are 2 types ->`
  1.Non_parameter functions
  2.parameter functions

def -> It is used for function

- `pass is a null statement or a placeholder that does nothing when executed. It is used when a statement is syntactically required but you don't want any code to run.`

Global vs Local variables

Feature	Local Variable	Global Variable
Scope	Inside a function or block	Throughout the entire program (module level)
Defined using	Inside a function	Outside all functions
Accessed using	Directly within the function	Directly anywhere, but needs `global` keyword to modify inside a function
Lifetime	Exists only during the function execution	Exists as long as the program runs
Modification inside function	Cannot be modified unless declared global	Can be modified using `global` keyword
Example	`def f(): x = 10` (x is local)	`x = 10` (defined outside any function)

Use case	Temporary calculations inside functions	Shared values across multiple functions

Feature	`*args`	`**kwargs`
Full Form	Arguments	Keyword Arguments
Use	To pass a variable number of **positional** arguments	To pass a variable number of **keyword** arguments
Syntax	`*args`	`**kwargs`
Data Type	Tuple	Dictionary
Access	Access elements by index	Access elements by key
Order Maintained	Yes (in order of arguments)	Yes (by keys)
Example Call	`func(1, 2, 3)`	`func(a=1, b=2)`
Example Definition	`def func(*args):`	`def func(**kwargs):`
Can Be Combined?	Yes, in function definitions like `def func(x, *args, **kwargs):`	Yes

---

return Value:

To let a function return a value, use the return statement:

pass Statement

function definitions cannot be empty, but if you for some reason have a function definition with no content, put in the pass statement to avoid getting an error.

Lambda Functions in Python A lambda function in Python is a small anonymous function defined using the lambda keyword. It's mostly used for short, throwaway functions.

Syntax: lambda arguments: expression

keyboard_arrow_down

Purpose:

The Purpose of Lambda functions is -> same like function only -> but there is no function name:

Lambda function -> A function without a function name

Lambda functions can take any number of arguments

Map | filter | reduce

Map = there is no need to use for loop because map will work like loop

syntax = map(function_name,iterables)

Feature	`map()`	`filter()`	`reduce()`
Purpose	Applies a function to **all items** in a list	Selects items that **match a condition**	Reduces a list to **a single cumulative value**
Returns	A new iterator with transformed values	A new iterator with only filtered values	A single result (not a list)
Function	Takes a function and a sequence	Takes a function (returns boolean) and a sequence	Takes a function and a sequence

Imported?	Built-in	Built-in	From `functools`: `from functools import reduce`
Example Use	Add 10 to each item	Keep only even numbers	Sum all numbers in a list

`map will take each value no need of for loop`

`filter will take only satisfied values`

Comprehensions in Python

Comprehensions are a concise way to create sequences (like lists, sets, or dictionaries) from existing iterables using a single line of code.

Intermediate Python Interview Questions with Answers

---

1. What is the difference between a for loop and a while loop in Python?

- A `for` loop is used when you know how many times you want to iterate.
- A `while` loop is used when the number of iterations is not known in advance and depends on a condition.

2. What is the use of the break statement in Python?

- The `break` statement is used to exit a loop prematurely when a certain condition is met.

3. What does the continue statement do?

- The `continue` statement skips the current iteration and moves to the next iteration of the loop.

4. What is the purpose of the pass statement?

- The `pass` statement does nothing and is used as a placeholder when a statement is syntactically required but you don't want to write any code yet.

5. What is the difference between if, elif, and else in Python?

- `if` checks the initial condition.
- `elif` checks other conditions if the previous ones were False.
- `else` runs if none of the `if` or `elif` conditions were True.

6. What is a function in Python?

- A function is a block of reusable code that performs a specific task.

7. What is the difference between parameters and arguments?

- Parameters are variables in a function definition; arguments are the actual values passed when calling the function.

8. What are *args and **kwargs?

- `*args` allows a function to accept any number of positional arguments.
- `**kwargs` allows a function to accept any number of keyword arguments.

9. What is a lambda function?

- A lambda function is an anonymous, one-line function defined using the `lambda` keyword.

10. How is a lambda function different from a normal function?

- Lambda functions are used for short, throwaway functions. They can have only one expression.

11. What are list comprehensions in Python?

- List comprehensions provide a concise way to create lists using a single line of code.

12. What are set and dictionary comprehensions?

- Similar to list comprehensions but create sets and dictionaries respectively.

13. How does the enumerate() function work in a for loop?

- `enumerate()` adds a counter to an iterable and returns it as an enumerate object.

14. What is the purpose of the range() function?

- `range()` generates a sequence of numbers, commonly used in for loops.

15. Can you use else with loops?

- Yes, the `else` block in a loop runs if the loop completes without encountering a `break` statement.

16. What is the scope of a variable?

- The scope determines where a variable can be accessed (local, global, nonlocal).

17. How do you define a default parameter in a function?

```
def greet(name="User"):
    print("Hello,", name)
```

18. Give an example of using map() with a lambda.

```
nums = [1, 2, 3]
squared = list(map(lambda x: x**2, nums))
print(squared)  # Output: [1, 4, 9]
```

19. What is the difference between is and == in Python?

- `is` checks for object identity.
- `==` checks for value equality.

20. What are list slicing and its syntax?

- List slicing is used to extract a portion of a list using `[start:stop:step]` syntax.