Tasks – in and not in, Combined with What You've Learned

Practice the **in** and **not in** operators with lists (and strings), then combine them with **conditions**, **loops**, **input**, **variables**, and **lists**. Create each file, run it, and check the output.

Run scripts with: `python3 script_name.py`

Part 1 – in and not in basics

Task 1.1 – in with a list (`in_list_basic.py`)

- Create `in_list_basic.py` . Assign `allowed = ["yes", "y", "ok"]` . Print the result of `"yes" in allowed` . Then print `"no" in allowed` . Use two print statements.

Expected output:

```
True
False
```

Task 1.2 – not in with a list (`notin_list_basic.py`)

- Create `notin_list_basic.py` . Assign `banned = ["spam", "bad", "x"]` . Read a word from the user with `input()` . If the word is **not in** the banned list (`if word not in banned:`), print "Allowed". Else print "Blocked". Use a variable for the word.

Expected output (example – input "hello"):

```
Enter word: hello
Allowed
```

Task 1.3 – in with numbers (`in_list_numbers.py`)

- Create `in_list_numbers.py` . Assign `numbers = [10, 20, 30, 40, 50]` . Ask the user for a number (int). If the number **is in** the list, print "In the list". Else print "Not in the list". Use a variable and if/else.

Expected output (example – 30 then 99):

```
Enter a number: 30
In the list
```

(Run with 99: "Not in the list".)

Task 1.4 – in with a string (substring) (`in_string.py`)

- Create `in_string.py` . In Python, **in** also works for strings: `"ab" in "abc"` is True. Assign `text = "hello world"` . Print the result of `"world" in text` . Then print the result of `"xyz" not in text` . Use variables or direct expressions.

Expected output:

```
True
True
```

Part 2 – in / not in with conditions and input

Task 2.1 – Valid choice from menu (`in_valid_choice.py`)

- Create `in_valid_choice.py` . Assign `options = ["1", "2", "3"]` . Ask the user "Choose 1, 2, or 3:" and read their input. If the input **is in** options, print "Valid: " + their choice. Else print "Invalid choice". Use a variable for the input and str() if needed.

Expected output (example – "2" then "9"):

```
Choose 1, 2, or 3: 2
Valid: 2
```

(Run with 9: "Invalid choice".)

Task 2.2 – Only allow if in whitelist (`in_whitelist.py`)

- Create `in_whitelist.py`. Assign `users = ["ali", "bo", "cat"]`. Ask for a username (input). If the username **is in** the users list, print "Welcome, [username]!". Else print "Unknown user". Use a variable. (Compare in lowercase if you want: `username.lower() in users` so "Ali" works when list has "ali".)

Expected output (example – "bo"):

```
Username: bo
Welcome, bo!
```

Task 2.3 – Block if in blacklist (`notin_blacklist.py`)

- Create `notin_blacklist.py`. Assign `blocked = ["admin", "root", "test"]`. Ask for a username. If the username is **not in** blocked, print "OK". Else print "Blocked". Use a variable and if/else.

Expected output (example – "john" then "admin"):

```
Username: john
OK
```

(Run with admin: "Blocked".)

Part 3 – in / not in with loops

Task 3.1 – Build list of items not in "seen" (`in_loop_filter.py`)

- Create `in_loop_filter.py`. Assign `all_items = [1, 2, 3, 4, 5]` and `seen = [2, 4]`. Create an empty list `new_items`. Loop over `all_items`: if the item is **not in** seen, append it to `new_items`. After the loop print `new_items`. Result: [1, 3, 5]. Use a for loop and if.

Expected output:

```
[1, 3, 5]
```

Task 3.2 – Check every input until "quit" in list (`in_loop_quit.py`)

- Create `in_loop_quit.py` . Assign `quit_words = ["quit", "exit", "q"]` . Use a loop that runs at most 5 times. Each time: read a word. If the word **is in** quit_words, print "Bye" and break. Else print "You said: " + word. Use a variable for the word and a for loop with range(5).

Expected output (example – hello, quit):

```
Word: hello
You said: hello
Word: quit
Bye
```

Task 3.3 – Count how many of a list are in another list (`in_loop_count.py`)

- Create `in_loop_count.py` . Assign `answers = [1, 2, 3, 2, 1]` and `correct = [1, 3]` . Use a variable count = 0 and a for loop over answers. For each answer, if it **is in** correct, add 1 to count. After the loop print "Score: " + str(count). Result: 3 (three answers are 1 or 3). Use variables and str().

Expected output:

```
Score: 3
```

Part 4 – Combined scenarios

Task 4.1 – Add only if not already in list (`in_combined_add_unique.py`)

- Create `in_combined_add_unique.py`. Start with an empty list `tags = []`. Ask "How many tags to add?" and read n. In a for loop (n times): read a tag (string). If the tag is **not in** tags, append it and print "Added". Else print "Already exists". After the loop print the list. Use variables. Test with e.g. 4 tags: python, code, python, code (python and code added once, then "Already exists" twice).

Expected output (example):

```
How many tags to add? 4
Tag 1: python
Added
Tag 2: code
Added
Tag 3: python
Already exists
Tag 4: code
Already exists
['python', 'code']
```

Task 4.2 – Menu: check choice with in (`in_combined_menu.py`)

- Create `in_combined_menu.py`. Assign `valid = ["a", "b", "c", "q"]`. Loop: print "Options: a, b, c, q (quit)", read choice. If choice **is in** valid: if choice == "q" print "Bye" and break; else print "You chose " + choice. If choice **not in** valid, print "Invalid". Use variables and in/not in.

Expected output (example – b, x, q):

```
Options: a, b, c, q (quit): b
You chose b
Options: a, b, c, q (quit): x
Invalid
Options: a, b, c, q (quit): q
Bye
```

Task 4.3 – Password: must contain a digit (in with string) (`in_combined_password.py`)

- Create `in_combined_password.py`. Assign `digits = "0123456789"`. Ask for a password (input). Use a variable found = False and a for loop over the password string (for char in password). If char **is in** digits, set found = True and break. After the loop: if found print "Password has a digit"; else print "Password must contain a digit". Use variables. (This combines in, string, loop, break, and condition.)

Expected output (example – "abc" then "ab3c"):

```
Password: abc
Password must contain a digit
```

(Run with "ab3c": "Password has a digit".)

Task 4.4 – List of allowed, loop until valid input (`in_combined_retry.py`)

- Create `in_combined_retry.py`. Assign `valid_yes = ["yes", "y"]` and `valid_no = ["no", "n"]`. Use a while True loop: ask "Continue? (yes/no):" and read answer (lowercase: answer.lower()). If answer **is in** valid_yes, print "Continuing" and break. If answer **is in** valid_no, print "Stopping" and break. Else print "Please enter yes or no" and loop again. Use variables and in.

Expected output (example – maybe, then yes):

```
Continue? (yes/no): maybe
Please enter yes or no
Continue? (yes/no): yes
Continuing
```

Task 4.5 – Two lists: items in both (intersection idea) (`in_combined_common.py`)

- Create `in_combined_common.py`. Assign `list1 = [1, 2, 3, 4, 5]` and `list2 = [3, 4, 5, 6, 7]`. Create an empty list `common`. Loop over list1: if the element **is in** list2, append it to common. After the loop print "Common: " + str(common). Result: [3, 4, 5]. Use a for loop and in.

Expected output:

```
Common: [3, 4, 5]
```

Done

You've used: **in** and **not in** with lists and strings; combined them with **if/else**, **loops**, **input**, **variables**, **break**, and **building lists**. Use in/not in for membership checks, validation, and filtering.