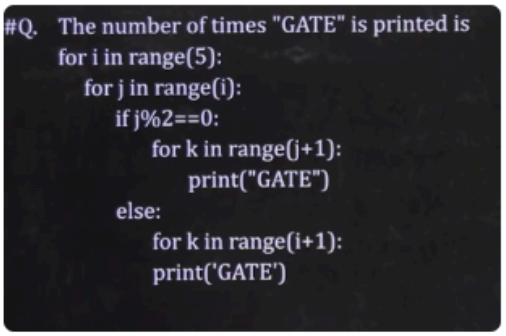


Let's carefully analyze the question step by step 👇

🧩 Code:

```python
for i in range(5):
    for j in range(i):
        if j % 2 == 0:
            for k in range(j + 1):
                print("GATE")
        else:
            for k in range(i + 1):
                print("GATE")
```

We need to count **how many times "GATE"** gets printed in total.

🧠 Step-by-step breakdown:

When `i = 0`
`range(i)` → `range(0)` → no iterations
✅ Prints = **0**

When `i = 1`

`range(1)` → `j = 0`

- `j = 0` → even → `range(j+1) = range(1)` → prints **1 time**

✅ Total so far = 1

When `i = 2`

`range(2)` → `j = 0, 1`

- `j = 0 (even)` → `range(1)` → prints **1**
- `j = 1 (odd)` → `range(i+1)=range(3)` → prints **3**

✅ Total this round = 1 + 3 = 4

Cumulative = 1 + 4 = **5**

When `i = 3`

`range(3)` → `j = 0, 1, 2`

- `j = 0 (even)` → `range(1)` → prints **1**
- `j = 1 (odd)` → `range(4)` → prints **4**
- `j = 2 (even)` → `range(3)` → prints **3**

✅ Total this round = 1 + 4 + 3 = 8

Cumulative = 5 + 8 = **13**

When `i = 4`

`range(4)` → `j = 0, 1, 2, 3`

- `j = 0 (even)` → `range(1)` → **1**
- `j = 1 (odd)` → `range(5)` → **5**
- `j = 2 (even)` → `range(3)` → **3**
- `j = 3 (odd)` → `range(5)` → **5**

✅ Total this round = 1 + 5 + 3 + 5 = 14

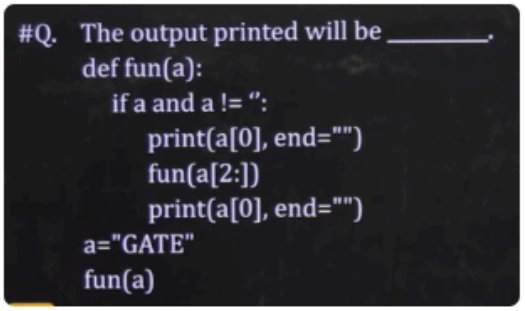
Cumulative = 13 + 14 = **27**

✅ **Final Answer:**

```bash
"GATE" is printed 27 times.
```

Copy code

Bad response



Let's carefully go step-by-step through this recursive function 👇

Code:

```python
def fun(a):
    if a and a != "":
        print(a[0], end="")
        fun(a[2:])
        print(a[0], end="")

a = "GATE"
fun(a)
```

Copy code

Step 1️⃣ — Initial Call

`a = "GATE"`

✅ `a[0] = 'G'` → print `'G'`
Now call `fun(a[2:])` → `fun("TE")`

Step 2️⃣ — Second Call (`a = "TE"`)

✅ `a[0] = 'T'` → print `'T'`
Now call `fun(a[2:])` → `fun("")`

Step 3️⃣ — Third Call (`a = ""`)

Condition `if a and a != ""` is **False**, so this returns without printing anything.

Now recursion starts **unwinding** 1️⃣

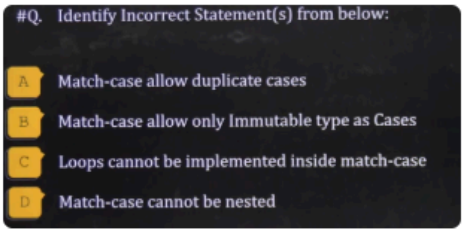
- From Step 2: after recursive call → `print(a[0])` again → print `'T'`
- From Step 1: after recursive call → print `'G'` again

📜 Printed order:

1️⃣ `'G'` (before recursion)
2️⃣ `'T'` (before recursion)
3️⃣ `'T'` (after recursion)
4️⃣ `'G'` (after recursion)

✅ Final Output:

```nginx
GTTG
```



Let's analyze each option carefully 👇

A. Match-case allow duplicate cases

❌ **Incorrect** —
Python **does not allow duplicate cases**.
Only the first matching `case` executes, and having duplicates is redundant (though technically not a syntax error, it's *logically invalid* — only the first one ever runs).
Hence, this statement is **incorrect**.

B. Match-case allow only Immutable type as Cases

✅ **Correct** —
Case patterns must be **immutable** (like numbers, strings, tuples, etc.), because mutable ones (like lists, dicts) are not hashable and cannot be matched reliably.

C. Loops cannot be implemented inside match-case

❌ **Incorrect** —
You **can absolutely use loops** (`for` , `while`) inside any `case` block — it's just normal Python code inside.
So this statement is **wrong**.

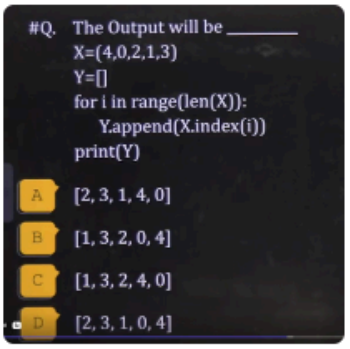
D. Match-case cannot be nested

❌ **Incorrect** —
You **can nest match-case** statements inside another — Python supports nesting, just like `if` statements.

✅ **Final Answer:**

Incorrect statements are:

```mathematica
A, C, and D
```



Let's carefully analyze the code from the image 👇

```python
X = (4, 0, 2, 1, 3)
Y = []

for i in range(len(X)):    # range(5) → [0, 1, 2, 3, 4]
    Y.append(X.index(i))

print(Y)
```

Step-by-step Execution 🧠

i	X.index(i)	Explanation
0	1	because 0 is at index 1 in X
1	3	because 1 is at index 3 in X
2	2	because 2 is at index 2 in X
3	4	because 3 is at index 4 in X
4	0	because 4 is at index 0 in X

✅ Output:

python Copy code

```
[1, 3, 2, 4, 0]
```

✅ Correct Option:

C → [1, 3, 2, 4, 0]

#Q. The final result printed will be _____

```
result=1
for i in range(5):
    result+=1
    for j in range(1,i+1):
        result+=j
        if j==i%j:
            break
        if i==result:
            continue
print(result)
```

Let's analyze this carefully 👇

Code:

python Copy code

```python
result = 1
for i in range(5):
    result += 1
    for j in range(1, i + 1):
        result += j
        if j == i % j:
            break
        if i == result:
            continue
print(result)
```

Let's **trace it step by step** 🧠

Initially:

`result = 1`

---

i = 0

- `result += 1` → result = 2
- Inner loop → `range(1, 0+1)` → `range(1,1)` → ❌ (no iteration)
- End of inner loop → result = 2

---

i = 1

- `result += 1` → result = 3
- Inner loop: `range(1, 2)` → j = 1
  - `result += j` → result = 4
  - `i % j = 1 % 1 = 0` → `j == 0?` ❌
  - `i == result?` → 1 == 4 ❌
- End of inner loop → result = 4

---

i = 2

- `result += 1` → result = **5**
- Inner loop: `range(1, 3)` → j = 1, 2

j = 1:

- `result += 1` → result = 6
- `i % j = 2 % 1 = 0` → `j == 0?` ❌
- `i == result?` → 2 == 6 ❌

j = 2:

- `result += 2` → result = 8
- `i % j = 2 % 2 = 0` → `j == 0?` ❌
- `i == result?` → 2 == 8 ❌
- Loop ends → result = **8**

---

i = 3

- `result += 1` → result = **9**
- Inner loop: `range(1, 4)` → j = 1, 2, 3

j = 1:

- result = 10
- `i % j = 3 % 1 = 0` → `j == 0?` ❌

j = 2:

- result = 12
- `i % j = 3 % 2 = 1` → `j == 1?` ❌

j = 3:

- result = 15
- `i % j = 3 % 3 = 0` → `j == 0?` ❌

Loop ends → result = **15**

---

i = 4

- `result += 1` → result = **16**

- Inner loop: `range(1, 5)` → j = 1, 2, 3, 4

j = 1:

- result = 17
- `i % j = 4 % 1 = 0` → ❌

j = 2:

- result = 19
- `i % j = 4 % 2 = 0` → ❌

j = 3:

- result = 22
- `i % j = 4 % 3 = 1` → ❌

j = 4:

- result = 26
- `i % j = 4 % 4 = 0` → ❌

Loop ends → result = **26**

✅ **Final Output:**

python Copy code

```
26
```

Answer:

✅ The final result printed will be **26**

how to solve this question fast

🔥 Great question — this is exactly the kind of Python logic puzzle where **pattern spotting and control flow shortcuts** save you massive time.

Let's break how to **think fast**, not just calculate. 👇

⚙️ Step-by-step Speed Strategy

🧩 Step 1: Understand the structure

The question has **two loops** and **some conditions**, like:

```python
for i in range(5):
    result += 1
    for j in range(1, i+1):
        result += j
        if j == i % j: break
        if i == result: continue
```

When you see:

- `result += 1` outside inner loop
- `result += j` inside inner loop
  it means **result increases cumulatively**, no reset.

🧠 Step 2: Identify where conditions actually trigger

`if j == i % j`
For small values, check quickly:

i	j	i % j	Condition True?
0	—	—	—
1	1	0	❌
2	1	0	❌
2	2	0	❌
3	1	0	❌
3	2	1	❌
3	3	0	❌
4	1	0	❌
4	2	0	❌
4	3	1	❌
4	4	0	❌

👉 **This never becomes True** because `i % j` is always less than `j` except when `i % j == 0`, and then `j != 0`.
So you can safely **ignore the** `break`.

🎹 Step 3: Simplify mentally

Now the code effectively becomes:

python Copy code

```python
result = 1
for i in range(5):
    result += 1
    for j in range(1, i+1):
        result += j
```

📝 Step 4: Spot the pattern

Compute quickly:

i	Outer +1	Inner Adds (sum of 1..i)	Cumulative Change
0	+1	0	+1
1	+1	1	+2
2	+1	3	+4
3	+1	6	+7
4	+1	10	+11

Now sum all these:
1 + 2 + 4 + 7 + 11 = **25**

Start was `result = 1` → 1 + 25 = **26**

✅ **Final result = 26**

⚡ Step 5: Shortcut formula (for similar ones)

If `result += 1 + (1+2+...+i)` inside a loop
→ Total addition after n iterations =

$$\sum_{i=0}^{n-1} \left(1 + \frac{i(i+1)}{2}\right)$$

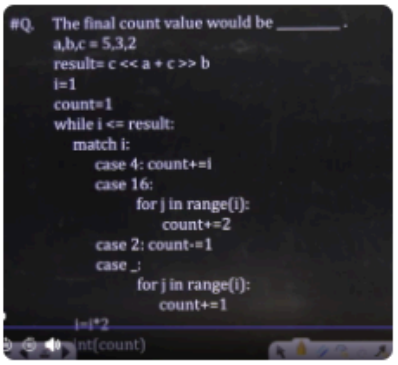
You can plug directly if needed.

⚡ TL;DR — Fast Mental Trick

When you see a problem like this:

1. **Check if any `break` or `continue` actually triggers.**
2. **Reduce it to clean arithmetic updates.**
3. **Spot small pattern (add +1 + sum of numbers).**
4. **Compute only totals — not every step-by-step.**

Would you like me to show a "**5-second method**" template (mental checklist) you can apply to *any* nested-loop-with-condition Python question like this (for exam speed)?



Let's solve it step by step 👇

🧩 Given Code

```python
a, b, c = 5, 3, 2
result = c << a + c >> b
i = 1
count = 1

while i <= result:
    match i:
        case 4:
            count += i
        case 16:
            for j in range(i):
                count += 2
        case 2:
            count -= 1
        case _:
            for j in range(i):
                count += 1
    i = i * 2

print(count)
```

🧮 Step 1: Evaluate `result`

Expression:

```python
result = c << a + c >> b
```

Operator precedence:
`+` → before `<<` / `>>`

So:

```r
a + c = 5 + 2 = 7
result = (c << 7) >> b
```

Now:

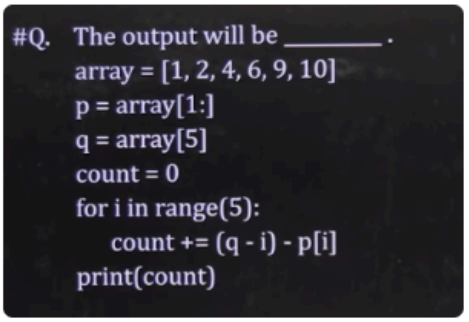
```
bash                                                    Copy code

c = 2
2 << 7 = 2 * 2^7 = 256
256 >> 3 = 256 / 2^3 = 32
```

✅ `result = 32`

🎼 Step 2: Trace the loop

Iteration	i	Condition (i ≤ 32)	Match case	Effect on `count`	i update
Start	—	—	—	count = 1	—
1	1	✅	`_`	`for j in range(1): count+=1` → +1 → count=2	i=2
2	2	✅	`case 2`	`count -= 1` → 2-1=1	i=4
3	4	✅	`case 4`	`count += i` → 1+4=5	i=8
4	8	✅	`_`	`for j in range(8): count+=1` → +8 → 13	i=16
5	16	✅	`case 16`	loop `for j in range(16): count+=2` → +32 → 45	i=32
6	32	✅	`_`	`for j in range(32): count+=1` → +32 → 77	i=64
7	64	❌	loop ends	—	—

✅ **Final value of `count` = 77**



Let's carefully go step by step 👇

🧩 Given code:

python Copy code

```python
array = [1, 2, 4, 6, 9, 10]
p = array[1:]
q = array[5]
count = 0

for i in range(5):
    count += (q - i) - p[i]

print(count)
```

Step 1️⃣ — Evaluate the variables

ini Copy code

```ini
array = [1, 2, 4, 6, 9, 10]
p = array[1:] = [2, 4, 6, 9, 10]
q = array[5] = 10
```

Step 2️⃣ — Loop details

`range(5)` → i = 0, 1, 2, 3, 4

Formula each time:

```makefile
count += (q - i) - p[i]
```

Now compute step by step 👇

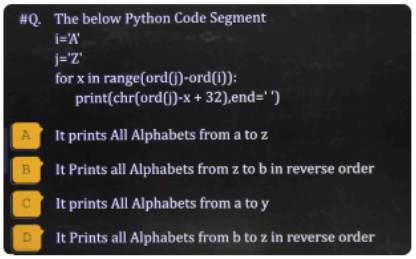
i	q-i	p[i]	(q - i) - p[i]	count
0	10	2	8	8
1	9	4	5	13
2	8	6	2	15
3	7	9	-2	13
4	6	10	-4	9

✅ Final Output:

```
9
```

✅ Answer:

Output → 9



Let's decode this question carefully 👇

🧩 Given Code

python Copy code

```python
i = 'A'
j = 'Z'

for x in range(ord(j) - ord(i)):
    print(chr(ord(j) - x + 32), end=' ')
```

Step 1️⃣ — Find ASCII values

perl Copy code

```perl
ord('A') = 65
ord('Z') = 90
```

So:

python Copy code

```python
range(ord(j) - ord(i)) = range(90 - 65) = range(25)
```

→ i.e. x = 0 to 24

Step 2️⃣ — Inside loop

Expression:

```python
chr(ord(j) - x + 32)
```

Let's compute a few values:

x	ord(j)-x	+32	chr(...)	Output
0	90	122	`'z'`	
1	89	121	`'y'`	
2	88	120	`'x'`	
3	87	119	`'w'`	
...	...	...	...	...
24	66	98	`'b'`	

✅ So it prints:

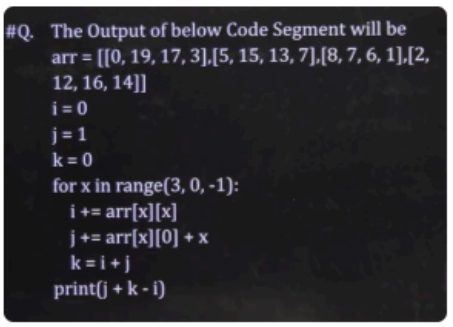
```css
z y x w ... c b
```

That is — **all alphabets from z to b in reverse order.**

✅ Final Answer:

Option B →

👉 *It prints all alphabets from z to b in reverse order.*



Let's carefully go step by step 👇

🧩 Code:

python Copy code

```python
arr = [[0, 19, 17, 3],
       [5, 15, 13, 7],
       [8, 7, 6, 1],
       [2, 12, 16, 14]]

i = 0
j = 1
k = 0

for x in range(3, 0, -1):
    i += arr[x][x]
    j += arr[x][0] + x
    k = i + j

print(j + k - i)
```

Step 1 — Analyze the loop

`range(3, 0, -1)` → [3, 2, 1]

We'll go through **x = 3, 2, 1**

Step 2 — Start values

```ini
i = 0
j = 1
k = 0
```

🔷 Iteration 1: x = 3

```makefile
i += arr[3][3] = 14 → i = 14
j += arr[3][0] + 3 = 2 + 3 = 5 → j = 6
k = i + j = 14 + 6 = 20
```

✅ After x=3 → i=14, j=6, k=20

🔷 Iteration 2: x = 2

```makefile
i += arr[2][2] = 6 → i = 20
j += arr[2][0] + 2 = 8 + 2 = 10 → j = 16
k = i + j = 36
```

✅ After x=2 → i=20, j=16, k=36

🔷 Iteration 3: x = 1

makefile

```
i += arr[1][1] = 15 → i = 35
j += arr[1][0] + 1 = 5 + 1 = 6 → j = 22
k = i + j = 57
```

✅ After x=1 → i=35, j=22, k=57

Step 3 — Final print:

bash

```
print(j + k - i)
= 22 + 57 - 35
= 44
```

✅ **Final Output:**

```
44
```