

6.2 Arrays, stacks and queues

Name: _____ Class: _____ Date: _____

Total: 24 marks

KEY WORDS static • dynamic • array • index
• two-dimensional array • stack • queue

Objective

Build the skills to answer exam questions on **data structures** – static and dynamic structures, one- and two-dimensional arrays, stacks and queues.

You must be able to:

- compare a **static** and a **dynamic** data structure
- read, change and total items in a one-dimensional **array** by **index**
- use `grid[row][col]` and nested loops on a **two-dimensional array**
- trace the operations on a **stack** (LIFO) and a **queue** (FIFO), and give a use of each

1 Key knowledge and worked examples

Study these first. Each one shows the method for a question type used later.

■ Key knowledge: static and dynamic

	Static	Dynamic
size	fixed when created	grows and shrinks at run time
memory	reserved in advance	used as needed
risk	full: overflow	runs out of memory

A Python list is dynamic: `append` adds an item at the end and `pop()` removes one.

■ Worked example: a two-dimensional array

	column				
	0	1	2	3	
row 0	4	7	1	9	
row 1	2	8	5	3	grid[1][2] == 5
row 2	6	0	7	4	row first, then column

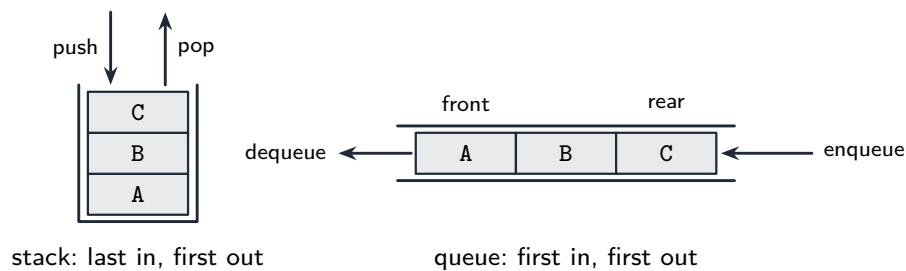
Find the total of column 3 of `grid`.

```
total = 0
for row in range(len(grid)):
    total = total + grid[row][3]
```

1. `len(grid)` is 3: the number of rows.
2. The column index stays at 3; the row index runs 0, 1, 2.
3. $9 + 3 + 4 = 16$.

To visit every cell, nest the loops: rows outside, columns inside.

■ Key knowledge: stacks and queues



	Stack	Queue
order	LIFO	FIFO
add	push on the top	enqueue at the rear
remove	pop from the top	dequeue from the front
uses	undo, back button	print jobs, key buffer

- Check `isEmpty` before removing: removing from an empty structure is an **under-flow**.

■ Worked example: tracing a stack

Start empty. `push(5)`, `push(8)`, `pop()`, `push(2)`, `push(7)`, `pop()`.

Operation	Stack (bottom → top)	Returned
<code>push(5)</code>	5	
<code>push(8)</code>	5, 8	
<code>pop()</code>	5	8
<code>push(2)</code>	5, 2	
<code>push(7)</code>	5, 2, 7	
<code>pop()</code>	5, 2	7

The same operations on a queue would remove 5, then 8: the oldest item leaves first.

2 Practice

2.1 `temps = [14, 17, 21, 19, 16]`. State: (i) `temps[1]`; (ii) `len(temps)`; (iii) the list after `temps[4] = 18`; (iv) the index of the largest value. [4]

2.2 Using the grid above, state: (i) `grid[2][0]`; (ii) `grid[0][3]`; (iii) the total of row 2. [3]

2.3 Starting with an empty queue: `enqueue("P")`, `enqueue("Q")`, `dequeue()`, `enqueue("R")`, `enqueue("S")`, `dequeue()`. State the item removed by each `dequeue` and the final queue from front to rear. [3]

2.4 Give **one** advantage of a static data structure and **one** of a dynamic data structure. [2]

3 Exam-style questions

3.1 Which data structure suits the back button of a web browser? [1]

- ☐ **A** a queue
B a stack
C a two-dimensional array
D a static one-dimensional array

3.2 `seats` is a 2D array with 10 rows and 20 columns. Which expression gives the last seat of the last row? [1]

- ☐ **A** `seats[10][20]`
B `seats[20][10]`
C `seats[9][19]`
D `seats[19][9]`

3.3 The numbers 1, 2, 3, 4 are pushed onto an empty stack in that order, then two items are popped. Which items are left, bottom to top? [1]

- ☐ **A** 1, 2
B 3, 4
C 2, 3
D 1, 4

3.4 A school stores weekly reading minutes for 3 students over 4 weeks in `mins`, one row per student.

```
mins = [[30, 45, 20, 50],  
        [60, 15, 40, 35],  
        [25, 55, 30, 10]]
```

(a) State the value of `mins[1][2]` and what it means. [2]

(b) Write Python code that prints the total minutes for each student. [4]

(c) State the output of your code. [1]

(d) A print server sends documents to one printer. Explain why the documents are held in a queue and not a stack. [2]

Solutions

Each answer shows the steps, then the **result**.

2.1

- (i) **17**
- (ii) **5**
- (iii) [**14**, 17, 21, 19, 18]
- (iv) 21 is at index **2**

2.2

- (i) **6**
- (ii) **9**
- (iii) $6 + 0 + 7 + 4 = \mathbf{17}$

2.3

- first **dequeue** removes **P**
- second **dequeue** removes **Q**
- final queue **R, S**

2.4

- static: e.g. memory is reserved in advance, so it cannot run out while running; direct, fast access by index
- dynamic: e.g. no wasted memory; it can grow to hold any number of items

3.1 B

- the last page visited is the first one returned to: LIFO

3.2 C

- indexes run from 0, so the last row is 9 and the last column 19

3.3 A

- 4 and then 3 are popped from the top

3.4

(a)

- **40**
- the minutes of the second student (row 1) in the third week (column 2)

(b) One answer:

```
for s in range(len(mins)):
    total = 0
    for w in range(len(mins[s])):
        total = total + mins[s][w]
    print(total)
```

- outer loop over the rows
- total reset to 0 for each student
- inner loop adds `mins[s][w]`
- prints once per student, after the inner loop

(c)

- **145, 150, 120**

(d)

- a queue is FIFO: documents print in the order they were sent
- a stack would print the latest first, so an early document could wait for ever