Exercise walk-through

6.2
Arrays, stacks and queues

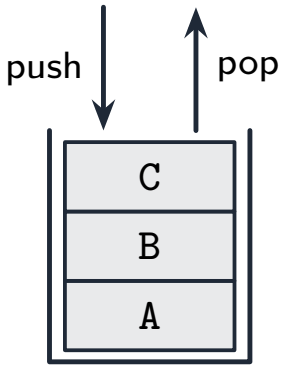
International Baccalaureate · IB Diploma · Computer Science · HL

Catching bad input with try and except

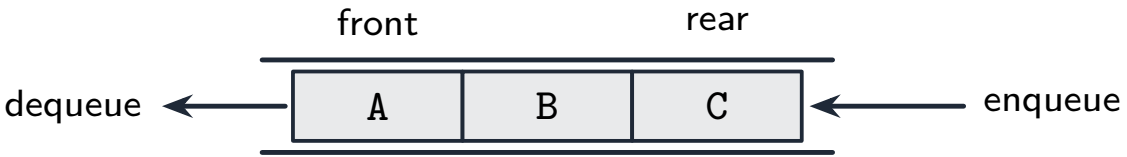
- An unhandled **exception** stops the program.
- The loop repeats until the cast succeeds.

```
while True:
    text = input("Quantity: ")
    try:
        qty = int(text)
        break
    except ValueError:
        print("Enter a whole number.")
```

Stacks and queues



stack: last in, first out



queue: first in, first out

> Stack: push and pop at the top (undo, back button). Queue: enqueue at the rear, dequeue at the front (print jobs).

Check yourself

Your turn

Exam habits to check before you answer.

A. Does your trace table give a new row each time a variable changes?

B. Did you state that Python indexes from 0?

C. Did you name a stack or queue's order, its two operations and a real use?

D. Did you open files with with and handle bad input with try and except?

E. (AHL) Did you name the base case, the recursive case and how each call moves towards it?

Exercises 2.1·2.2

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]

Exercises 2.1·2.2

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]

2.1

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

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

2.2

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

Exercises 2.3·2.4

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]

Exercises 2.3·2.4

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]

2.3

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

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

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

Exercises 3.1·3.2

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]`

Exercises 3.1·3.2

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

3.1 B

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

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]`

3.2 C

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

Exercise 3.3

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

Exercise 3.3

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

3.3 A

- 4 and then 3 are popped from the top

Exercise 3.4 (a)

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]

Exercise 3.4 (a)

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]

(a)

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

Exercise 3.4 (b)

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]]
```

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

Exercise 3.4 (b)

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]]
```

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

(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

Exercise 3.4 (c)

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]]
```

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

Exercise 3.4 (c)

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]]
```

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

(c)

- **145**, **150**, **120**

Exercise 3.4 (d)

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]]
```

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

Exercise 3.4 (d)

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]]
```

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

(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