Programming

IB DP Computer Science B2 Programming full package (AHL recursion included)

Why does a working program fail on Monday?

> Good programs are written for every input they might meet, not only the ones the programmer tried.

- Theme B2 in Python; AHL adds recursion.
- Prerequisites: B1 decomposition, abstraction and pseudocode.

Data types and casting

Type	Python	Example	Use
integer	int	42	counts, indexes
real	float	3.75	measurements
Boolean	bool	True	flags, conditions
string	str	"Ada"	text

Use the evidence

int("42") casts text to a number. 7 / 2 is 3.5; 7 // 2 is 3; 7 % 2 is 1.

Strings are indexed from 0

- Slices stop before the end **index**.
- Strings are **immutable**.

```
code = "ib-2027"
len(code)         # 7
code[0]           # "i"
code[3:7]         # "2027"
code.upper()      # "IB-2027"
code.find("-")    # 2; -1 if not found
"2027" in code    # True
```

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.")
```

Three kinds of error

Kind	**When it shows**	**Example**
syntax error	before the program runs	a missing colon after if x $>$ 0
runtime error	while it runs, as an **exception**	an **index** past the end of a list
logic error	never; the output is wrong	$<$ where $<=$ was meant

Tracing a loop

k	k % 2 == 0	total
		0
1	False	0
2	True	2
3	False	2
4	True	6

Use the evidence

Loop: for k in range(1, 5): if k % 2 == 0: total = total + k. Output 6.

Data types

integer	whole number: 7
real	decimal: 3.5
char	one letter: 'A'
string	text: "hello"
boolean	true or false

> The type sets what operations are allowed

A trace table

count	total	output
1	5	
2	12	
3	20	20

> Trace by hand before you run it

Static and dynamic structures

Static (array in Java or C)

Size fixed; memory reserved in advance, perhaps wasted.

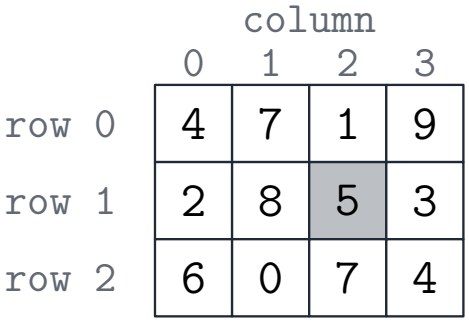
Fast direct access by **index**; **overflow** when full.

Dynamic (Python list)

Grows and shrinks at run time; memory used as needed.

May need extra work to grow; risk of running out of memory.

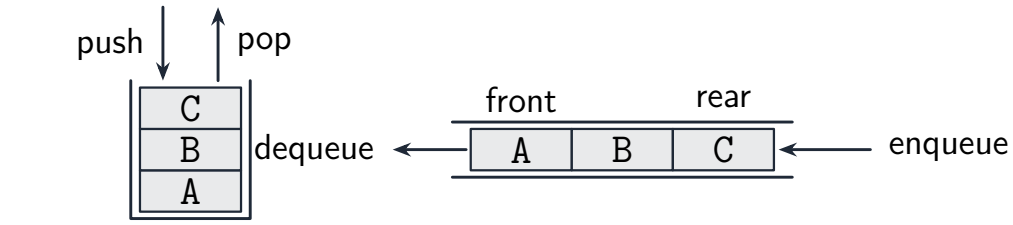
Two-dimensional arrays



`grid[1][2] == 5`
row first, then column

grid[1][2] is row 1, column 2: the value 5.

Stacks and queues



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

Three constructs and three loops

Loop	Python	Use when
count-controlled	for i in range(n):	the number of repeats is known
condition-controlled	while condition:	repeat until something happens
over a collection	for item in items:	visit each item once

Use the evidence

A while loop whose condition never becomes false is an **infinite loop**.

Functions and their contracts

- total is local; it exists only during the call.
- mean([]) breaks the contract.

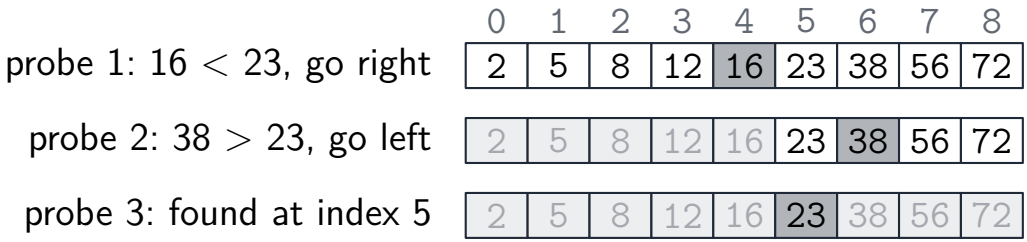
```
def mean(values):
    """Return the mean of a non-empty list of numbers."""
    total = 0
    for v in values:
        total = total + v
    return total / len(values)

print(mean([4, 8, 9]))   # 7.0
```

Big O growth

Big O	**Example**	**Steps for n = 1000**
$O(1)$	reading marks[2]	1
$O(\log n)$	**binary search**	about 10
$O(n)$	**linear search**	1000
$O(n^2)$	bubble or **selection sort**	about 1 000 000

Binary search, probe by probe



Find 23: mid 4 (16, go right), mid 6 (38, go left), mid 5: found, **return** 5.

Bubble sort and selection sort

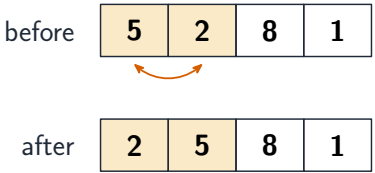
After pass	Bubble sort	Selection sort
1	[1, 4, 2, 5]	[1, 5, 4, 2]
2	[1, 2, 4, 5]	[1, 2, 4, 5]
3	no swaps: stop	[1, 2, 4, 5]

Use the evidence

Sorting [5, 1, 4, 2]. **Bubble sort** swaps neighbours; **selection sort** swaps the smallest into place.

Bubble sort

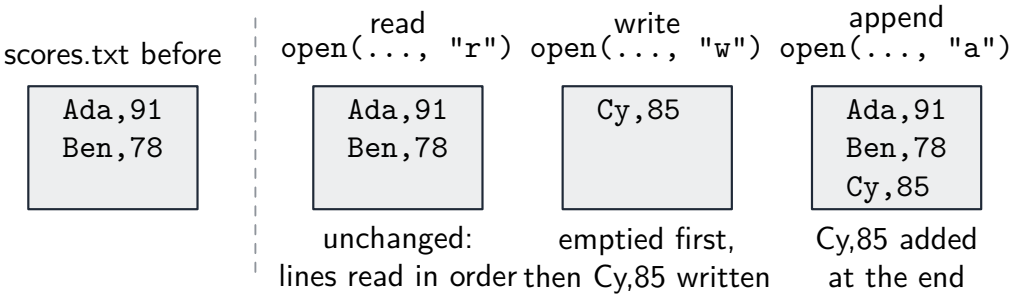
compare the pair 5 and 2: $5 > 2$, so swap them



then repeat for each pair, until no swaps are needed

> Repeat passes until no swaps are made

File modes



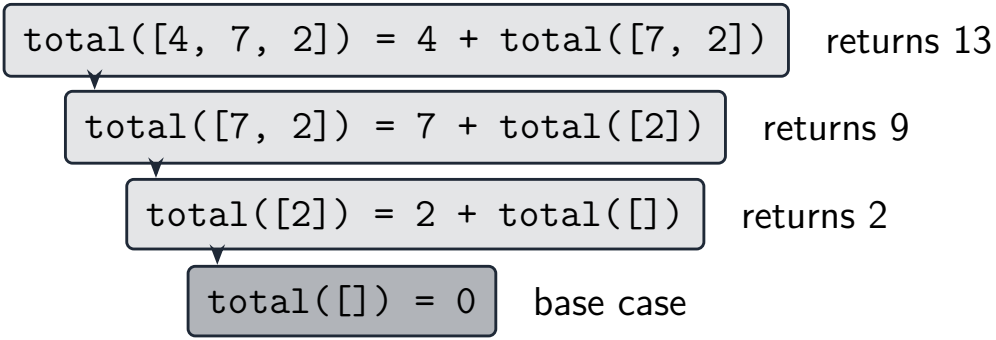
"r" reads from the start, "w" empties then writes, "a" adds to the end.

Reading a file line by line

- strip() removes the newline.
- split(",") breaks a line into fields.

```
best_name, best = "", -1
with open("scores.txt", "r") as f:
    for line in f:
        name, score = line.strip().split(",")
        if int(score) > best:
            best_name, best = name, int(score)
print(best_name, best)
```

(AHL) A function that calls itself



Calls go down to the **base case**; results come back up, last called first finished.

(AHL) Recursion or iteration?

	Recursion	**Iteration**
code	often shorter, matches the definition	often longer
memory	one stack frame per call	one set of variables
risk	**stack overflow** if too deep	an **infinite loop**
natural for	trees, nested folders, **quicksort**	counting, running totals

Practice

Your turn

Work these, then check the handout answers.

A. s = "Programming": give len(s), s[3:7] and s.find("g").

B. Push 10, 20, 30; pop; push 40. What is on top?

C. Which values does a binary search for 31 compare in [3, 8, 12, 19, 25, 31, 40]?

D. Show [6, 3, 8, 2] after each bubble sort pass.

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?