Exercise walk-through

6.5 Recursion

International Baccalaureate · IB Diploma · Computer Science · HL

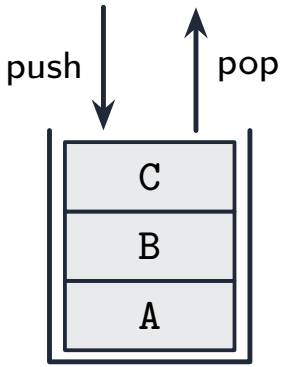
Tracing a loop

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

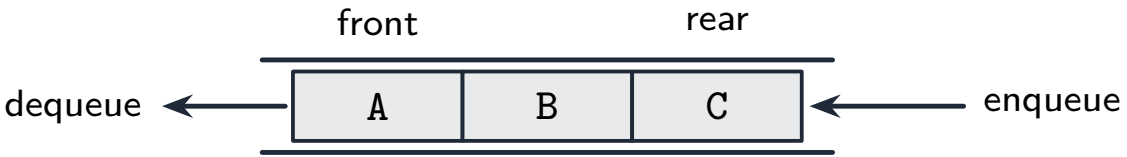
Use the evidence

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

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).

(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

Exercises 2.1·2.2

2.1 For `total` above, state (i) the base case; (ii) how each call moves towards it. [2]

2.2 Trace `digits(9)` and `digits(250)`, giving the number of calls and the value returned for each. [2]

Exercises 2.1·2.2

2.1 For `total` above, state (i) the base case; (ii) how each call moves towards it. [2]

2.1

- (i) the empty list `[]`, which returns 0
- (ii) each call passes `items[1:]`, one item shorter

2.2 Trace `digits(9)` and `digits(250)`, giving the number of calls and the value returned for each. [2]

2.2

- `digits(9)`: **1** call, returns **1**
- `digits(250)` → `digits(25)` → `digits(2)`: **3** calls, returns **3**

Exercise 2.3

2.3 `def fact(n): return n * fact(n - 1)` has no base case. Write a corrected version for $n \geq 0$, given that $0! = 1$. [2]

Exercise 2.3

2.3 `def fact(n): return n * fact(n - 1)` has no base case. Write a corrected version for $n \geq 0$, given that $0! = 1$. [2]

2.3

```
def fact(n):
    if n == 0:
        return 1
    return n * fact(n - 1)
```

- base case `n == 0` returning 1
- recursive case unchanged, after the base case

Exercise 3.1

3.1 What is returned by `f(3)`? [1]

```
def f(n):
    if n == 0:
        return 0
    return n + f(n - 1)
```

- ☐ **A** 3
- ☐ **B** 5
- ☐ **C** 6
- ☐ **D** 9

Exercise 3.1

3.1 What is returned by `f(3)`? [1]

```
def f(n):
    if n == 0:
        return 0
    return n + f(n - 1)
```

- ☐ **A** 3
- ☐ **B** 5
- ☐ **C** 6
- ☐ **D** 9

3.1 C

- $3 + 2 + 1 + 0 = 6$

Exercise 3.2

3.2 Why does a deep recursion use more memory than the equivalent loop? [1]

- [ ] **A** each call adds a frame to the call stack
- [ ] **B** recursive functions cannot use local variables
- [ ] **C** the base case is stored on disk
- [ ] **D** recursion copies the whole program

Exercise 3.2

3.2 Why does a deep recursion use more memory than the equivalent loop? [1]

- ☐ **A** each call adds a frame to the call stack
- **B** recursive functions cannot use local variables
- **C** the base case is stored on disk
- **D** recursion copies the whole program

3.2 A

- each waiting call keeps its frame

Exercise 3.3

3.3 Which input makes g run until the stack overflows? [1]

```
def g(n):
    if n == 1:
        return 1
    return g(n - 2)
```

- ☐ **A** 1
- ☐ **B** 3
- ☐ **C** 4
- ☐ **D** 5

Exercise 3.3

3.3 Which input makes g run until the stack overflows? [1]

```
def g(n):
    if n == 1:
        return 1
    return g(n - 2)
```

- ☐ **A** 1
- ☐ **B** 3
- ☐ **C** 4
- ☐ **D** 5

3.3 C

- 4, 2, 0, -2, …: it steps over 1 and never reaches the base case

Exercise 3.4 (a)

3.4 A function counts the vowels in a word.

```
def vowels(w):
    if w == "":
        return 0
    if w[0] in "aeiou":
        return 1 + vowels(w[1:])
    return vowels(w[1:])
```

(a) Identify the base case. [1]

Exercise 3.4 (a)

3.4 A function counts the vowels in a word.

```
def vowels(w):
    if w == "":
        return 0
    if w[0] in "aeiou":
        return 1 + vowels(w[1:])
    return vowels(w[1:])
```

(a) Identify the base case. [1]

(a)

- `w == ""`, returning 0

Exercise 3.4 (b)

3.4 A function counts the vowels in a word.

```
def vowels(w):
    if w == "":
        return 0
    if w[0] in "aeiou":
        return 1 + vowels(w[1:])
    return vowels(w[1:])
```

(b) Trace `vowels("tea")`, giving each call and the value it returns. [4]

Exercise 3.4 (b)

3.4 A function counts the vowels in a word.

```
def vowels(w):
    if w == "":
        return 0
    if w[0] in "aeiou":
        return 1 + vowels(w[1:])
    return vowels(w[1:])
```

(b) Trace `vowels("tea")`, giving each call and the value it returns. [4]

(b)

Call	First letter	Returns
`vowels("tea")`	`t`: not a vowel	`vowels("ea")` = 2
`vowels("ea")`	`e`: vowel	1 + `vowels("a")` = 2
`vowels("a")`	`a`: vowel	1 + `vowels("")` = 1
`vowels("")`	base case	0

- calls go down to `vowels("")`
- base case returns 0
- values 1, 2, 2 passed back up

- result **2**

Exercise 3.4 (c)

3.4 A function counts the vowels in a word.

```
def vowels(w):
    if w == "":
        return 0
    if w[0] in "aeiou":
        return 1 + vowels(w[1:])
    return vowels(w[1:])
```

(c) State how many calls are made in total for a word of n letters. [1]

Exercise 3.4 (c)

3.4 A function counts the vowels in a word.

```
def vowels(w):
    if w == "":
        return 0
    if w[0] in "aeiou":
        return 1 + vowels(w[1:])
    return vowels(w[1:])
```

(c) State how many calls are made in total for a word of n letters. [1]

(c)

- $n + 1$: one per letter, plus the empty string

Exercise 3.4 (d)

3.4 A function counts the vowels in a word.

```
def vowels(w):
    if w == "":
        return 0
    if w[0] in "aeiou":
        return 1 + vowels(w[1:])
    return vowels(w[1:])
```

(d) Write an iterative version of `vowels`. [3]

Exercise 3.4 (d)

3.4 A function counts the vowels in a word.

```
def vowels(w):
    if w == "":
        return 0
    if w[0] in "aeiou":
        return 1 + vowels(w[1:])
    return vowels(w[1:])
```

(d) Write an iterative version of `vowels`. [3]

(d) One answer:

```
def vowels(w):
    count = 0
    for ch in w:
        if ch in "aeiou":
            count = count + 1
    return count
```

- loop over each character
- test against the vowels
- count and return after the loop

Exercise 3.4 (e)

3.4 A function counts the vowels in a word.

```
def vowels(w):
    if w == "":
        return 0
    if w[0] in "aeiou":
        return 1 + vowels(w[1:])
    return vowels(w[1:])
```

(e) Give **one** reason the iterative version is better for a word of a million letters. [1]

Exercise 3.4 (e)

3.4 A function counts the vowels in a word.

```
def vowels(w):
    if w == "":
        return 0
    if w[0] in "aeiou":
        return 1 + vowels(w[1:])
    return vowels(w[1:])
```

(e) Give **one** reason the iterative version is better for a word of a million letters. [1]

(e)

- a million calls would overflow the call stack; the loop uses one set of variables