6.5 Recursion

Name: _____ Class: _____ Date: _____ **Total: 19 marks**

KEY WORDS base case • recursive case • recursive function • call stack • stack overflow • iteration

Objective

Build the skills to answer HL exam questions on **recursion** – base and recursive cases, tracing recursive calls, the call stack, and recursion compared with iteration.

You must be able to:

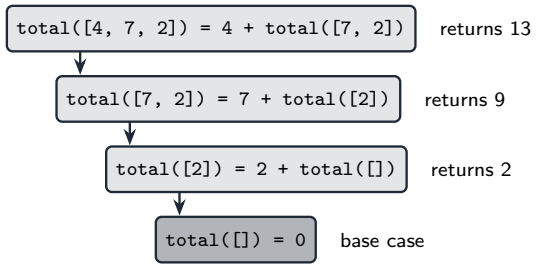
- identify the **base case** and the **recursive case** of a **recursive function**
- trace a recursive call, giving each call and the value it returns
- explain the role of the **call stack** and the cause of a **stack overflow**
- compare recursion with **iteration**

1 Key knowledge and worked examples

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

■ Key knowledge: the two parts

```
def total(items):
    if items == []:                    # base case
        return 0
    return items[0] + total(items[1:])   # recursive case
```



- Each call waits on the call stack, with its own local variables, until the call it made returns.
- The calls finish in reverse order: the base case first, the first call last.
- With no base case, or a step that never reaches it, the stack fills: a stack overflow (`RecursionError` in Python).

■ Worked example: tracing with a table

```
def digits(n):
    if n < 10:
        return 1
    return 1 + digits(n // 10)
```

Trace `digits(4071)`.

Call	`n < 10?`	Returns
`digits(4071)`	no	`1 + digits(407)` = 4
`digits(407)`	no	`1 + digits(40)` = 3
`digits(40)`	no	`1 + digits(4)` = 2
`digits(4)`	yes: base case	1

1. Go **down** the table, making each call, until the base case.
2. Work **up** the Returns column, filling in each waiting call.
3. Result **4**, from four calls: `n // 10` shrinks `n` towards the base case.

■ Key knowledge: recursion or iteration?

	Recursion	Iteration
code	short, follows the definition	often longer
memory	one stack frame per call	one set of variables
failure	stack overflow	infinite loop

Recursion suits problems made of smaller copies of themselves: trees, nested folders, **quicksort** .

2 Practice

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]

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]

3 Exam-style questions

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

3.2 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

3.3 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

3.4 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]

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

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

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

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