

Solutions

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

2.1

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

2.2

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

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

3.1 C

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

3.2 A

- each waiting call keeps its frame

3.3 C

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

3.4

(a)

- `w == ""`, returning 0

(b)

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

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

- result **2**

(c)

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

(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

(e)

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