COMP 102 · FORMAN CHRISTIAN UNIVERSITY

# Midterm Exam

Programming Fundamentals

Fakhir Shaheen · Sections A and B · Fall 2026

Tue 20 Oct 2026
1 h 45 min
100 marks
13 pages

**YOU**

Name

Roll number

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |

Section ◯ A ◯ B

*I pledge that the work in this booklet is my own, done without help from anyone or anything the rules do not allow.*

Signature

**FOR THE EXAMINER**

| Ch. | Out of | Score |
|---|---|---|
| 1 | 20 | |
| 2 | 20 | |
| 3 | 15 | |
| 4 | 10 | |
| 5 | 15 | |
| 6 | 20 | |
| **Total** | **100** | |
| Bonus | 10 | |

**THE RULES**

**Closed** Do not open this booklet until you are told to.

**Phones** Phones and smart watches switched off and in your bag. No books, notes or calculators.

**Pages** Check that you have all 13 pages. The last two pages are for rough work.

**Python** All code is Python 3 as Thonny runs it. Where a line stops with an error, write `Error`.

**Time** Each chapter shows its marks and about how many minutes it needs. Stuck? Move on.

**Your code** Indentation counts. Use only what the lectures so far have taught.

**Doubts** If a question seems wrong, write what you assumed and carry on. Invigilators cannot explain the content.

**Answers** Write each answer on its rule or in its box. Work elsewhere is not marked unless you point to it.

**Pen** Answers in pen; pencil only for rough work.

**Roll no** Put your roll number at the top of every page.

# 1 Inside the Shell

20 marks · about 18 min

*For every line, write exactly what Python shows. If a line stops with an error, write the word* `Error`*.*

(a) **[8 marks] A Shell session.** On each blank, write what the Shell shows after that line is entered. A line that only stores a value shows nothing, so it has no blank.

```
>>> runs = 112
>>> runs // 10                          _____
>>> runs % 10                           _____
>>> runs / 2                            _____
>>> flight = "PK-3417"
>>> len(flight)                         _____
>>> flight[1] + flight[3]               _____
>>> flight[3:5]                         _____
>>> int(flight[3:7]) + 1                _____
>>> flight[0] == "P" and runs > 112     _____
```

(b) **[4 marks] One program, four runs.** The program reads a distance in km and prints how to travel. The user runs the program **four separate times**, and each time types a different number: one row of the table for each run. Write what the program prints each time.

```
km = int(input())
if km < 5:
    print("walk")
elif km <= 20:
    print("cycle")
else:
    print("drive")
```

| The user types | The program prints |
|---|---|
| 4 | _____ |
| 5 | _____ |
| 20 | _____ |
| 21 | _____ |

(c) **[8 marks] A second Shell session.** The file in the small box has just been run with F5, so the Shell knows `share`. Further down, `goals` holds the goals scored in four matches.

```
def share(sweets, kids=2):
    return sweets // kids
```

```
>>> share(90)                                    _____
>>> share(90, 20)                                _____
>>> twice = lambda n: n * 2
>>> twice(share(10)) + 1                         _____
>>> goals = [3, 0, 5, 2]
>>> goals[1] + goals[-1]                         _____
>>> goals[1:3]                                   _____
>>> sum(goals) / len(goals)                      _____
>>> low, high = min(goals), max(goals)
>>> high - low                                   _____
>>> goals[4]                                     _____
```

## 2 Follow the Tracks

20 marks · about 20 min

*Go through each program one step at a time and write down what is in memory.*

(a) **[8 marks] A trace table.** To *trace* a program, follow it one line at a time and keep track of its variables. One run of the loop body (lines 4 to 6) is one **round**, and each row of the table is one round.

```
1 money = 90
2 week = 0
3 while money > 15:
4     spent = money // 3
5     money = money - spent
6     week = week + 1
7 print(week, money)
```

| Round | spent | money | week |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |

i. **[5 marks]** For each round, write the values of `spent`, `money` and `week` at the **end** of that round.

ii. **[1 mark]** The program prints: _____

iii. **[2 marks]** Line 3 is changed to `while money >= 12:` and the program is run again. Now it prints: _____

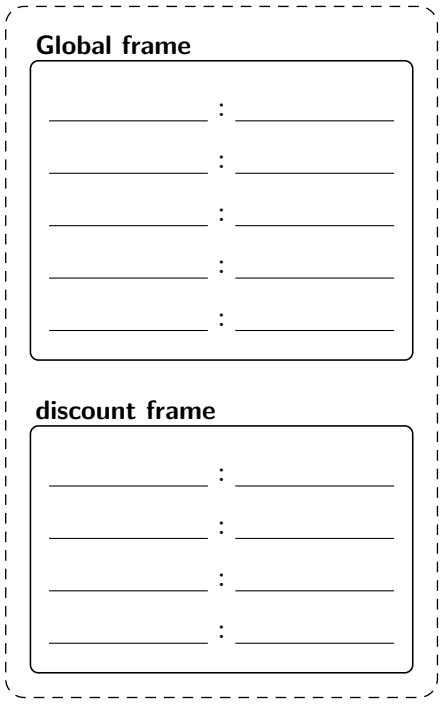
(b) **[7 marks] Frames.** Fill in the frames the way the lectures draw them: one `name : value` row for each name, and for a name that holds a function write `func` as its value. Leave spare rows empty. If a frame no longer exists at that moment, cross out its whole box. Memory is drawn at **two moments** of the run. Each dashed box is one moment, a snapshot: both frames inside it belong to that moment.

```
1 rate = 80
2 def discount(price, rate):
3     off = price * rate // 100
4     # * pause here
5     return off
6 price = 120
7 n = discount(price - 20, 75)
```

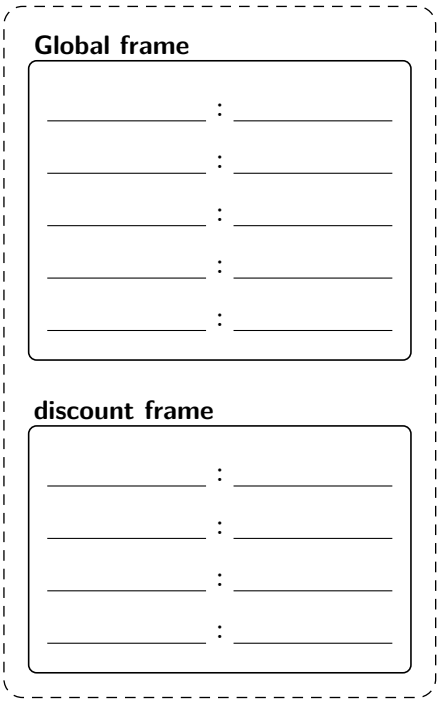
i. **[5 marks] Snapshot 1:** the program has stopped at the line marked `*`, inside the call, just before line 5 runs. Fill in its two frames.

ii. **[2 marks] Snapshot 2:** line 7 has finished. Fill in its two frames.

**Snapshot 1**
stopped at `*`, just before line 5 runs

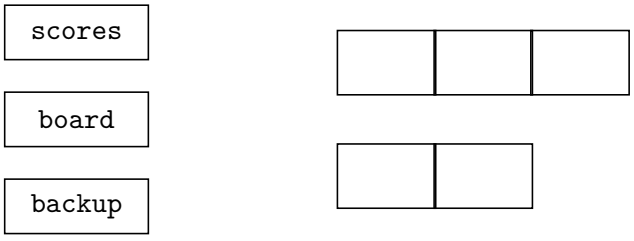



**Snapshot 2**
after line 7 has finished




(c) **[5 marks] Arrows.** Read the program on the left.

```
1 scores = [112, 98]
2 board = scores
3 backup = scores[:]
4 board.append(105)
5 backup[0] = 0
```




i. **[2 marks]** The picture shows memory after line 5: three names and two list objects. Draw an arrow from each name to the list it points at.

ii. **[2 marks]** Write the value held in each of the five cells.

iii. **[1 mark]** Write `True` or `False` on each rule.

`scores is board` _____ `scores == backup` _____

## 3 The Case of the Wrong Count

15 marks · about 16 min

*One program gives the wrong answer. For another, work out what the user typed from what it printed.*

(a) **[8 marks] Two bugs.** `count_top(total, got)` gets two lists of the same length: `total[i]` is the total marks of test `i`, and `got[i]` is the student's mark in it. A test is a *top* test if the student got 80% marks or more. Line 9 should print `2`: the student got 80% in the first test (top), 50% in the second and 90% in the third (top). It does not. The program has exactly **two** bugs. No line is missing and no line is extra.

```
1 def count_top(total, got):
2     """How many tests are top tests."""
3     top = 0
4     for i in range(len(total)):
5         rate = got[i] * 100 // total[i]
6         if rate > 80:
7             top = top + 1
8         return top
9 print(count_top([100, 120, 50], [80, 60, 45]))
```

What a **correct** `count_top` returns for two other calls:

```
>>> count_top([50], [20])
0
>>> count_top([10, 5], [9, 5])
2
```

i. **[2 marks]** As it stands, line 9 prints _____ and line 5 runs _____ time(s).

ii. **[6 marks]** Give the number of each line that has a bug, and write the line as it should be. If only the indentation of a line is wrong, say which line it should line up with.

| Line | The line as it should be |
|---|---|
| | |
| | |

(b) **[7 marks] Work backwards.** This program has **no bugs**. The user types the number of days, then the minutes of exercise for each day, one per line. No number is negative.

```
1 days = int(input())
2 total = 0
3 best = 0
4 for i in range(days):
5     minutes = int(input())
6     total = total + minutes
7     if minutes > best:
8         best = minutes
9 print(total, best)
```

i. **[2 marks]** The user types the four lines in the box, exactly as shown.

```
3
90
120
90
```

The program prints: _____

ii. **[2 marks]** In another run the program printed `250 100`. The user typed `3` first, and two of the days had 100 and 60 minutes. How many minutes did the third day have? _____

iii. **[1 mark]** The shortest input that makes the program print `0 0` is: _____

iv. **[2 marks]** Someone says that a run printed `100 120`. In one sentence, explain why this program can never print that.

_____

_____

## 4 Three Answers, One Job

10 marks · about 12 min

*Three students wrote the same function. Only one of them is right. Run each one in your head and find it.*

**What was asked for.** The function `clean(counts)` is given a list of numbers. It **returns a new list** that holds only the numbers that are not `0`, in the same order. **The list it was given must not change.**

**The test.** The test is the code in the box on the **right**. Each answer below is run with it. With a correct `clean`, the test prints `[98]` and then `[0, 0, 98, 0]`.

**The test**

```
votes = [0, 0, 98, 0]
r = clean(votes)
print(r)
print(votes)
```

(a) [**5 marks**] Beside each answer, write **everything** the test prints when it is run with that answer, one line of output on each rule. Leave spare rules empty. (Answer A: 2 marks, B: 1 mark, C: 2 marks.)

**Answer A**

```
def clean(counts):
    kept = []
    for c in counts:
        if c != 0:
            kept.append(c)
    print(kept)
```

The test prints:

_____

_____

_____

**Answer B**

```
def clean(counts):
    kept = []
    i = 0
    while i < len(counts):
        if counts[i] > 0:
            kept = kept + [counts[i]]
        i = i + 1
    return kept
```

The test prints:

_____

_____

_____

**Answer C**

```
def clean(counts):
    for c in counts:
        if c == 0:
            counts.remove(c)
    return counts
```

The test prints:

_____

_____

_____

(b) **[1 mark]** Which answer does what was asked for? Answer _____

(c) **[2 marks]** For each of the other two answers, say in one sentence what is wrong with it.

_____

_____

_____

_____

(d) **[1 mark]** Answer A can be made correct by changing one line. Write the new line.

_____

(e) **[1 mark]** After the test with Answer C, is `r is votes` `True` or `False`? _____

---

## 5 Lost Lines

15 marks · about 15 min

*One program has its lines mixed up. Another program has gaps. Put them back together.*

(a) **[7 marks] Shuffled lines.** The days of a year are numbered 1 to 365. `due_day(start, wait)` returns the day that comes `wait` days after day `start`. If the caller does not give `wait`, it is 60. After day 365 the count starts again from 1.

The eight lines in the box are that program with its lines shuffled. Each line is shown with its correct indentation. **One line does not belong to the program.**

```
A       return day
B   for start in [10, 320]:
C           day = day - 365
D   def due_day(start, wait=60):
E       print(start, "->", due_day(start))
F       day = start + wait
G           day = 365 - day
H       if day > 365:
```

When the seven lines are in the right order and the program is run, it prints **exactly this**:

```
10 -> 70
320 -> 15
```

Each line shows a start day, then the day 60 days after it.

i. **[6 marks]** Write the letters of the seven lines of the program in the order they must be typed, the first line in the first box.

first [ ] [ ] [ ] [ ] [ ] [ ] [ ] last

ii. **[1 mark]** The letter of the line that does not belong: _____

(b) **[8 marks] Fill the gaps.** The program reads the runs of some cricket players. The first line of the input is the number of players. Then comes one line for each player: a name, a space, and the runs. For a player with runs, the program prints the name and one star for every 20 runs (only full 20s count). A player with 0 runs goes into the list `zeros`. At the end the program prints the total runs and that list.

**The user types**

```
3
Ali 104
Sara 0
Omar 65
```

**The program prints**

```
Ali *****
Omar ***
Total: 169
Zeros: ['Sara']
```

**Write the missing code for each blank on the rule below with the same letter.** One mark a blank.

```
n = ___ⓐ___
total = 0
zeros = ___ⓑ___
for i in ___ⓒ___:
    parts = input().___ⓓ___
    runs = ___ⓔ___
    total = total + runs
    if ___ⓕ___:
        zeros.___ⓖ___
    else:
        print(parts[0], ___ⓗ___)
print("Total:", total)
print("Zeros:", zeros)
```

ⓐ _____ ⓑ _____ ⓒ _____

ⓓ _____ ⓔ _____

ⓕ _____ ⓖ _____

ⓗ _____

## 6 Write It Yourself

20 marks · about 22 min

*Each part tells you what a function must do and gives you an empty box. Write the code yourself.*

Major marks are for the logic. Indentation is part of the logic, so make it clear which lines are inside which. Each function must **return** its answer, not print it.

(a) **[8 marks] The average.**

Write the function `how_hot(temps)`. `temps` is a list of one or more temperatures. Find their **average**. If it is below 28, return `'cool'`. If it is 31 or more, return `'hot'`. Anything in between returns `'warm'`.

```
>>> how_hot([27, 28, 26])
'cool'
>>> how_hot([30, 32, 31])
'hot'
>>> how_hot([28, 28])
'warm'
```

(b) **[3 marks] A change of rule.** Change your `how_hot` from part (a): before the average is found, the **hottest** and the **coolest** temperature are left out. (`temps` now always has three or more temperatures.) For example, with `[27, 28, 35, 26]` the 35 and the 26 are left out, so the average is of 27 and 28, which is 27.5, and the answer is `'cool'`.

In the box, write **only the lines that change**, as they should be now. Do not copy the whole function again.

(c) **[5 marks] The first hot day.** Write the function `first_hot(temps, limit)`. `temps` has one temperature for each day: `temps[0]` is day 1, `temps[1]` is day 2, and so on. A day is **hot** if its temperature is `limit` or more. Return the **day number** of the first hot day. If no day is hot, return `0`.

```
>>> first_hot([27, 29, 31, 30, 32], 31)
3
>>> first_hot([33, 20], 31)
1
>>> first_hot([27, 29], 31)
0
```

First call: days 1 and 2 are not hot; day 3 (31) is the first hot day.

Second call: day 1 (33) is already hot.

Third call: no day is hot.

(d) **[4 marks] The longest hot spell.** Write the function `hot_spell(temps, limit)`. A day is **hot** if its temperature is `limit` or more. A **hot spell** is a group of hot days that come one straight after another. Return the number of days in the **longest** hot spell. If no day is hot, return `0`.

```
>>> hot_spell([31, 32, 29, 31, 33, 34, 30], 31)
3
>>> hot_spell([31, 20, 31], 31)
1
>>> hot_spell([27, 29], 31)
0
```

First call: 31, 32 is a spell of 2 days; 29 is not hot; then 31, 33, 34 is a spell of 3 days. The longest is 3.

Second call: two spells of 1 day each.

Third call: no day is hot.

# 7 Bonus: Look Closer

10 bonus marks · about 10 min

(a) **[5 bonus marks] The mystery list.**

```
1  def mystery(rows):
2      first = rows[0]
3      rows = rows[:]
4      rows.append(first)
5      first.append(len(rows))
6      return rows
7
8  a = [[1], [2]]
9  b = mystery(a)
10 b[0].append(0)
11 print(a)
12 print(b)
```

i. **[2 bonus marks]** Line 11 prints:

_____

ii. **[2 bonus marks]** Line 12 prints:

_____

iii. **[1 bonus mark]** After line 12, how many **different** list objects are there in memory? Count every list, inside and outside. _____

(b) **[5 bonus marks] The secret function.**

```
1  def secret(n):
2      result = 0
3      while n > 0:
4          result = result * 10 + n % 10
5          n = n // 10
6      return result
```

i. **[1 bonus mark]** `secret(1205)` returns:

_____

ii. **[1 bonus mark]** A number `n` with **three** digits for which `secret(n) == n` is:

_____

iii. **[1 bonus mark]** In one sentence, what does `secret` do to a number greater than 0?

_____

_____

iv. **[2 bonus marks]** `secret(secret(120))` returns _____, not 120. In one sentence, explain why.

_____

_____

End of the paper. Check that your roll number is on every page.

## Rough work

Nothing on this page is marked.

## Rough work

Nothing on this page is marked.