

To be completed by the professor			
	Program: BAM	Academic Year: 2026-2027	Semester: 1
	Course: Computer Programming I		
	Program Year: 1	Group: 1	Campus: Madrid
	Type of exam: Quiz ☐ Midterm ☒ Final ☐ Retake ☐		
	Professor: Daniel Precioso Garcelán		Date: 19/10/2026
	Total time allowed: 80 minutes		Num. Pages: 10

Student Name:	Student ID:
To be completed by the student	

DO NOT TURN THIS PAGE UNTIL THE PROFESSOR INSTRUCTS YOU TO DO SO.

EXAM RULES AND GUIDELINES

The following items are permitted on your desk:

Your ID, pens or pencils, erasers, and calculator (*only if approved by your professor*).

- All **electronic devices** (phones, smartwatches, headphones, etc.) must be **switched off and stored away**.
- **Food and drinks are not permitted** in the examination room.
- All **unauthorised materials** (including bags, notes, books, electronic devices, etc.) must be placed at the **front of the room**.
- If you are **unsure** whether an item is permitted, check with your professor before the examination begins.

During the Exam

- **Communication of any kind with other students is prohibited** (including talking, making gestures, passing notes, etc.).
- **Do not access any unauthorised materials or devices.**
- If you have a **question, raise your hand** (please note that this may not be allowed in some exams.)
- **Students are not permitted to leave and re-enter the examination room under any circumstances.** If you leave the room, you will not be permitted to return and continue the examination.
- **All students must remain in the examination room for at least the first 30 minutes.** A quiet and respectful environment must be maintained at all times.

After the Exam

- Ensure that **your name** is clearly written on your examination paper before submitting it.
- **Do not speak until you have left the examination room.**
- Once you have submitted your examination paper, remain silent to avoid disturbing other students.
- **Leave the examination room quietly and in an orderly manner** when instructed to do so by the professor.
- **Students are not permitted to remove the examination paper from the room.**

ACADEMIC INTEGRITY at IE: Cheating, plagiarism, impersonation, falsification, or any other form of academic misconduct constitutes a violation of the Student Code of Conduct and the examination regulations. Such violations may result in failing the examination or forfeiting the examination attempt. Repeated or serious misconduct may lead to further disciplinary action, including expulsion from the programme.

Student's signature _____

All examination papers and any additional materials used during the examination **MUST BE SECURELY STAPLED** together before submission.

Instructions

Always assume each code is run independently, so variables do not persist between exercises.

Questions 1-2: Open Problems (20 points each)

These are open-ended tasks. Explain your approach, outline algorithmic steps, and optionally provide compact working code snippets.

Questions 3-5: Debugging (10 points each)

These exercises show code cells with one or more errors. You have to find these errors, explain why they happen, and suggest a fix to them.

Questions 6-11: Multiple Choice (5 points each)

Circle the correct answer for each question, as each question has **only one** correct answer. Provide reasoning for your answer in the space next to each question. A correct answer with a sound explanation earns 5 points. An incorrect answer, faulty or missing reasoning, or no answer earns zero points.

Oral Justification. Students may be asked to provide oral justification for any of the exercises after completing the written portion.

1. You are designing a new line for the Madrid Metro. The goal is to select stations that provide connections to as many existing lines as possible. Your budget allows you to include exactly six stations.

You receive information about each station, including its name, and the Metro lines that serve it. For example:

- Manuel Becerra. Lines: 2 and 6.
 - Sol. Lines: 1, 2, and 3.
 - Plaza de Castilla. Lines: 1, 9, and 10.
 - and so on for all stations in the city.
- (a) Which Python data structure, or combination of data structures, would you use to store this information? Briefly justify your choice. *(5 points)*
- (b) At each step, select the unchosen station that serves the greatest number of lines not served by the stations already selected. Break ties by choosing the station whose name comes first in alphabetical order. Design a Python function that receives the station data and returns the six selected station names in the order chosen. *(15 points)*

Answer:

(a) We need to link every station with the lines that serve it, so the best option is a dictionary. The keys are the station names (strings), and the values are the lines of that station. Because the lines of a station have no order and are never repeated, each value is best stored as a set, e.g. `{"Sol": {1, 2, 3}}`. Sets are the right choice because we can later merge the lines of several stations with a union, compare them with a difference, and repeated lines disappear automatically. *(+5 points)*

(b) The function receives the dictionary and returns the six station names in selection order, for instance in a list. *(+2 points)*

Before the main loop, we initialize an empty list `chosen` for the selected stations, and an empty set `covered` for the lines already connected. *(+3 points)*

We need to repeat the selection six times, so we use a for loop, `for step in range(6)`. Inside it, a second (nested) loop goes through all the stations. We skip the ones already in `chosen`. For each remaining station, we count how many of its lines are new, using the set difference `stations[name].difference(covered)` and `len()`. *(+5 points)*

We keep track of the best station so far, and its number of new lines. Whenever a station provides more new lines than the best one so far, it replaces it. To break ties alphabetically, visit station names in alphabetical order and replace the current best only when a station provides strictly more new lines. If it provides fewer or the same number, keep the current best. *(+2 points)*

After the nested loop finishes, we add the best station to `chosen`, and update `covered` with the union of its lines. Using the updated set is what makes the next step ignore the lines we already have. After the main loop, we return `chosen`. *(+3 points)*

Comment: This greedy approach is not guaranteed to give the best possible group of six. We are not asking for that in this exercise.

(Optional, code)

```
1 def best_stations(stations: dict) -> list:
2     # stations example: {"Sol": {1, 2, 3}, "Manuel Becerra": {2, 6}}
3     chosen = []
4     covered = set()
5     # Choose six stations, one at a time
6     for step in range(6):
7         best_name = ""
8         best_new = -1
9         # Look at every station that is not chosen yet
10        for name in sorted(stations.keys()):
11            if name in chosen:
12                continue
13            # Count the lines that we do not connect yet
14            new_lines = len(stations[name].difference(covered))
15            if new_lines > best_new:
16                best_name = name
17                best_new = new_lines
18        # Add the best station, and its lines
19        chosen.append(best_name)
20        covered = covered.union(stations[best_name])
21    return chosen
```

2. You are analysing customer reviews and want to know how many times a given word is mentioned in a review. The search must ignore uppercase letters and punctuation, and it must only count **whole words**. For instance, the word “great” is mentioned three times in:

“Great food! Not so great service, but a GREAT price... greatly recommended.”

The three matches are “Great”, “great” and “GREAT”. The word “greatly” does not count. Treat punctuation as a word separator, so punctuation attached to a word does not change that word, and punctuation between letters separates them.

Design a Python function that gets a review and a target word as input, and returns the number of mentions. It must work for any text and any word, including a target word with punctuation around it. You may assume the target word contains no internal punctuation. (10 points)

Once your function is ready, you have a list with a thousand reviews and a set with a hundred target words. You want to know how many times each word is mentioned in total. How would you use your function to get this information? (10 points)

Answer:

First task. The function receives two strings, the review and the target word, and returns an integer. The review has uppercase letters and punctuation attached to the words (“Great”, “food!”, “price...”), so we first need to clean it. (+2 points)

First, we turn all letters into lowercase using `str.lower()`. To remove other special characters, we can use `str.replace()` to replace them with spaces. This way, any kind of punctuation is treated as a word separator. (+2 points)

Next, we turn the string into a list of words using `str.split()`. This gives us a list of clean words, all in lowercase, with no punctuation. (+2 points)

In order to count the target word, we also turn it into lowercase, and then we loop through the list of words, counting how many times it appears. We will need to initialize a counter to 0 and increment it for each match. (+2 points)

Finally, we return the count. (+2 points)

(Optional, code)

```
1 def count_mentions(review: str, word: str) -> int:
2     # Clean the review: lowercase and remove punctuation
3     review = review.lower()
4     for char in ". , ! ? ; : ( ) [ ] { } \ " ' " :
5         review = review.replace(char, " ")
6     # Split into words
7     words = review.split()
8     # Count the mentions of the target word
9     count = 0
10    for w in words:
11        if w == word.lower():
12            count += 1
13    return count
```

Second task. The target words are given as a set. We want one total per word, so the best structure is a dictionary, where the keys are the target words and the values are the total mentions. We initialize it with every word at 0. (+3 points)

We need to look at every review and, inside each of them, every target word. This is a nested loop: the outer loop goes through the reviews, and the inner loop goes through the words. (+3 points)

Inside the inner loop, we call our function with the current review and word, and add the result to the dictionary: `totals[word] += count_mentions(review, word)`. In total, the function is called 1000×100 times. At the end we have the totals for all the words. (+4 points)

Comment: It would be more efficient to clean each review only once, instead of once per word. This is optional. Calling the function 100,000 times is accepted.

(Optional, code)

```
1 def total_mentions(reviews: list[str], words: set[str]) -> dict:
2     # Start every target word at 0
3     totals = {}
4     for word in words:
5         totals[word] = 0
6     # For every review, check every target word
7     for review in reviews:
8         for word in words:
9             totals[word] += count_mentions(review, word)
10    return totals
```

3. The following function is intended to check if a number is prime. We tested it against the first ten numbers and got the following:

- True for 3, 5, 7, 9
- False for 1, 4, 6, 8, 10
- None for 2

As you can see, 9 is incorrectly classified and 2 is not classified at all. Find the errors in the code and explain why they occur and how to fix them.

```
1 def is_prime(n: int) -> bool:
2     if n <= 1:
3         return False
4     for i in range(2, n):
5         if n % i == 0:
6             return False
7     else:
8         return True
```

Answer:

There are two errors, and both come from the `return` statements inside the loop. First, the `else: return True` is inside the for loop, so the function returns `True` after checking only the first divisor (`i = 2`). That is why every odd number, like 9, is classified as prime. A number is only prime after **all** the divisors have been checked. (+5 points)

Second, for `n = 2`, `range(2, 2)` is empty, so the loop never runs and the function reaches the end without any `return`. In Python, that returns `None`. We fix both errors by removing the `else` and adding `return True` after the loop, outside of it. (+5 points)

(Optional, fixed code)

```
1 def is_prime(n: int) -> bool:
2     if n <= 1:
3         return False
4     for i in range(2, n):
5         if n % i == 0:
6             return False
7     return True
```

4. You are working on a library app. The dictionary `DICT_LOANS` keeps track of the number of times each book has been borrowed. Books with several parts are stored using a tuple (`title`, `part`) as the key. The function `register_loan` is intended to add one loan to a given book, creating the book with one loan if it is not in the dictionary yet. However, each of the two calls below, when run on its own, raises an error. Find the errors in the code and explain why they occur and how to fix them.

```
1 DICT_LOANS = {
2     "Dune": 12,
3     "Emma": 7,
4     "Ulysses": 3,
5     ("Harry Potter", "Part 1"): 18,
6     ("Harry Potter", "Part 2"): 15
7 }
8
9 def register_loan(title: str):
10     loans = DICT_LOANS.get(title)
11     DICT_LOANS[title] = loans + 1
12
13
14 register_loan("Matilda")
15 # TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
16
17 register_loan("Harry Potter", "Part 1")
18 # TypeError: register_loan() takes 1 positional argument but 2 were given
```

Answer:

There are two different errors. The first one is in line 10. The method `get()` returns `None` when the key is not in the dictionary, and "Matilda" is not there yet. Then line 11 tries to do `None + 1`, which is not a valid operation. We fix it by giving `get()` a default value as its second argument, `DICT_LOANS.get(title, 0)`. If the book does not exist, the result is 0, we add 1, and line 11 creates the new key with the value 1. An if-clause checking `title` in `DICT_LOANS` is also a valid fix. (+5 points)

The second error is in the call. The function receives only one argument, `title`, but the call passes two. The keys of the multi-part books are tuples, so the call must pass a single tuple: `register_loan(("Harry Potter", "Part 1"))`. Note the double parentheses: the inner ones create the tuple. The type hint `title: str` does not stop a tuple from being used, as type hints are not enforced. An alternative fix is to change the function to receive two arguments and to build the tuple inside it. (+5 points)

(Optional, fixed code)

```
1 def register_loan(title: str | tuple):
2     loans = DICT_LOANS.get(title, 0)
3     DICT_LOANS[title] = loans + 1
4
5
6 register_loan("Matilda")
7 register_loan(("Harry Potter", "Part 1"))
```

5. The following code is intended to convert all the temperatures in the list `ls_celsius`, which are strings such as "18C", into strings in Fahrenheit rounded to the nearest integer, such as "64F". However, it raises an error on line 5. If we remove "8C" from the list, that error disappears, but then the code never finishes. Find the errors in the code and explain why they happen and how to fix them.

```
1 ls_celsius = ["18C", "21C", "-5C", "30C", "8C"]
2
3 for temp in ls_celsius:
4     # Remove the "C" and convert to int
5     temp_num = int(temp[:2]) # ValueError: invalid literal for int() with base 10: '8C'
6     # Convert to Fahrenheit and round to the nearest integer
7     temp_fahrenheit = round(temp_num * 9 / 5 + 32)
8     # Save the new value as a string
9     ls_celsius.append(f"{temp_fahrenheit}F")
10
11 # Expected: ["64F", "70F", "23F", "86F", "46F"]
```

Answer:

The first error is the slice `temp[:2]` in line 5. It keeps the first two characters, which works for "18C" (it returns "18") but not for "8C", which has only one digit: the slice returns "8C", and `int()` fails because of the letter. The slice should remove the last character instead, `temp[:-1]`. Using `temp.replace("C", "")` or `temp.strip("C")` is also valid. (+5 points)

The second error is that we modify the list while we loop through it. Every iteration appends a new element at the end of `ls_celsius`, so the list is always longer than the number of iterations done. The loop never reaches the end of the list, and runs forever. In addition, `append()` adds the new values at the end instead of replacing the old ones, so even without the infinite loop the list would not end up as expected. We fix it by creating a new list and appending to it, or by looping over the indices, `range(len(ls_celsius))`, and overwriting each position. (+5 points)

(Optional, fixed code)

```
1 ls_celsius = ["18C", "21C", "-5C", "30C", "8C"]
2
3 # Option 1: new list
4 ls_fahrenheit = []
5 for temp in ls_celsius:
6     temp_num = int(temp[:-1])
7     temp_fahrenheit = round(temp_num * 9 / 5 + 32)
8     ls_fahrenheit.append(f"{temp_fahrenheit}F")
9
10 # Option 2: overwrite each position of the same list
11 for i in range(len(ls_celsius)):
12     temp_num = int(ls_celsius[i][:-1])
13     temp_fahrenheit = round(temp_num * 9 / 5 + 32)
14     ls_celsius[i] = f"{temp_fahrenheit}F"
```

Alternatively, we can use a list comprehension to create a new list. It does not modify the original list while iterating over it:

```
1 ls_fahrenheit = [f"{round(int(t[:-1]) * 9 / 5 + 32)}F" for t in ls_celsius]
```

6. What will be printed after running the following code?

```
1 secret = "$01_E23_L45_L67_089"
2 deciphered = secret[::4]
3 deciphered[0] = "H"
4 print(deciphered)
```

- A. "\$ELLO"
- B. "HELLO"
- C. "H2469"
- D. The code will raise an error.

7. Which one of the following objects can be used as a **key** of a dictionary?

- A. None of these choices can be used as a key.
- B. The set {"Arthur", "King"}
- C. The dictionary {"book": "Hyperion"}
- D. The list ["Nature", "Volume 1"]

8. What will be printed after running the following code?

```
1 name = "Rachel"
2 friend = "Phoebe"
3
4 def greet(name: str) -> None:
5     print("Hello, " + name + "!")
6     name = "Monica"
7
8 greet(friend)
```

- A. Hello, Monica!
- B. Hello, Rachel!
- C. Hello, Phoebe!
- D. The code will raise an error.

9. What happens when the following code is run and the user enters “abc123” (the right password)?

```
1 password = "abc123"
2
3 while True:
4     guess = input("Enter password: ") # abc123
5     if guess == password:
6         print("Welcome")
7     else:
8         print("Try again")
9         continue
10    break
```

- A. The console prints “Welcome” and the loop ends.
- B. The code will raise an error.
- C. The console starts printing “Welcome” repeatedly (never ends).
- D. The console prints “Welcome” and asks for the password again (never ends).

10. Which one of the following is an example of a **recursive function**?

- A. `fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)`
- B. `sum_list(lst)` using a while loop to iterate through the list.
- C. `factorial(n)` using a for loop with `range(1, n+1)`.
- D. `is_even(x)` being defined inside another function `is_prime(n)`.

11. After running this code, what will be printed?

```
1 team = ["Ana", "Luis"]
2 roster = (team, "Madrid")
3
4 team.append("Eva")
5 roster[0].remove("Ana")
6 print(team, "--", roster)
```

- A. `['Ana', 'Luis', 'Eva'] -- ([ 'Luis', 'Eva'], 'Madrid')`
- B. `['Ana', 'Luis', 'Eva'] -- ([ 'Luis'], 'Madrid')`
- C. The code will raise an error.
- D. `['Luis', 'Eva'] -- ([ 'Luis', 'Eva'], 'Madrid')`

Correct answers for Questions 6–11:

Question	Key
6	D
7	A
8	C
9	A
10	A
11	D

Cheatsheet

Strings:

- `s.lower()` returns a lowercase copy of `s`.
- `s[start:stop:step]` returns a slice. The `stop` index is excluded.
- `s.split()` splits at whitespace. `s.strip()` removes leading and trailing whitespace, while `s.strip(chars)` removes those characters instead.
- `s.replace(old, new)` returns a copy with occurrences of `old` replaced by `new`.
- `char.isalnum()` tests whether a character is a letter or digit.
- `int(s)` converts a string of digits into an integer, and fails otherwise. `round(x)` rounds `x` to the nearest integer.
- `f"{x}"` creates a string where `{x}` is replaced by the value of `x`. Text can be added before or after `{x}`.
- `s.count(pattern)` counts non-overlapping occurrences of `pattern` in `s`.

Lists and sets:

- `len(x)` returns the number of elements. `x in ls` tests whether `x` is an element of `ls`.
- `ls.append(x)` adds `x` to the end of a list.
- `ls.remove(x)` removes the first element equal to `x` from the list, and fails if there is none.
- `sorted(x)` returns a new sorted list with the elements of `x`.
- `set(ls)` creates a set from the elements of `ls`.
- `a.union(b)` and `a.difference(b)` return new sets.

Dictionaries:

- `d[key] = value` adds a key or replaces its value. `d[key]` retrieves the value for `key`.
- `d.get(key)` returns `None` when the key is absent. `d.get(key, default)` returns `default` instead.
- `key in d` tests whether `key` is a key in `d`. `d.keys()` gives the dictionary's keys.

Functions and control flow:

- `return` exits a function. A function that reaches its end without a `return` statement returns `None`.
- `break` exits the nearest enclosing loop. `continue` moves to the next iteration of the nearest enclosing loop.

Other:

- `range(start, stop)` generates integers from `start` up to, but not including, `stop`.
- `print(a, b)` prints values separated by spaces. `type(x)` returns the type of `x`.
- `input(prompt)` shows `prompt`, waits for the user to type, and returns what was typed as a string.