Name: _____
Surname: _____
Date: _____

Intructions

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

Questions 1-6: Multiple Choice (5 points each)

Circle the correct answer for each question, as each question has **only one** correct answer. A correct answer awards 5 points. There is no penalty for wrong choices.

Questions 7-9: 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 10-11: Open Problems (20 points each)

In these exercises, you will encounter problems that are more open-ended and may not have a single correct answer. Your task is to explain how you would approach each problem by breaking it down into smaller, manageable steps.
Describe how you would structure your code, which logical steps are needed, and what kind of Python tools or constructs (e.g., loops, conditionals, functions, lists, etc.) you would use. You are not required to write full code, but doing so is encouraged if it helps clarify your ideas.

---

1. What is the length of this set? {`"a", "a", "A", "b", "b", "bb", "c", "C"`}

   (a) 3
   (b) 5
   (c) 6
   (d) 8

2. Can I include a dictionary inside another dictionary?

   (a) Yes, always.
   (b) Yes, but only as a key.
   (c) Yes, but only as a value.
   (d) No, never.

3. What will be the last value of `x` after executing the following code?

```
x = 10

while x > 4:
    x = x // 2
```

   (a) 2
   (b) 4
   (c) 5
   (d) 10

4. What will be the output of the following code?

```
p = print

def func(a, b=2, c=3):
    p(a, b, c)

func(1, c=4)
```

   (a) `1 2 3`
   (b) `1 2 4`
   (c) `None`
   (d) Error

5. What will be the output of the following code?

```
print([0, 1, 2, (0, 1, 2), "abc"][3][1:])
```

   (a) `(0, 1, 2)`
   (b) `(1, 2)`
   (c) `[1, 2, "abc"]`
   (d) `[1, 2, (0, 1, 2), "abc", 3]`

6. What will be the output of the following code?

```
my_tuple = (1, 2)
my_list = list(my_tuple)
my_dict = {"a": my_list, "b": my_tuple}
my_list.append(3)
print(my_dict)
```

   (a) `{'a': [1, 2], 'b': (1, 2)}`
   (b) `{'a': [1, 2], 'b': (1, 2, 3)}`
   (c) `{'a': [1, 2, 3], 'b': (1, 2)}`
   (d) `{'a': [1, 2, 3], 'b': (1, 2, 3)}`

Answers: 1-c, 2-c, 3-a, 4-b, 5-b, 6-c

7. The following function is meant to round a floating point number to the nearest integer. However, it does not work as expected. Identify the issue and provide a corrected version of the function.

```
1  def custom_round(num: str) -> int:
2      """This function takes any number, turns it into a string,
3      and rounds it to the nearest integer without using the
4      built-in round function."""
5      if "." in num:
6          integer, decimals = num.split(".")
7          if int(decimals[0]) >= 5:
8              return int(integer) + 1
9          else:
10             return int(integer)
11     else:
12         return int(num)
13
14
15 print(custom_round(3.14159))
16 # Expected: 3
17 # Output: TypeError: argument of type 'float' is not iterable
```

Answer:

The function is assuming that the type hint in line 1 will turn any input argument into a string. This is wrong: type hints do not change the types of variables. *(+5 points)*

We need to add the following before line 5: `num = str(num)` *(+5 points)*

(Optional, fixed code)

```
1  def custom_round(num: str) -> int:
2    """This function takes any number, turns it into a string,
3    and rounds it to the nearest integer without using the
4    built-in round function."""
5    num = str(num)
6    if "." in num:
7      integer, decimals = num.split(".")
8    if int(decimals[0]) >= 5:
9      return int(integer) + 1
10   else:
11     return int(integer)
12   else:
13     return int(num)
```

8. This code is part of a game where a player can either take a certain amount of money or double it and pass it to the next player. However, the message always offer 1 euro. In addition, after the user inputs "take it", the program keeps playing. Identify the problems and suggest a fix.

```
1  value = 1
2  msg = f"I offer you {value} euros"
3
4  while True:
5    print(msg)  # This is always "I offer you 1 euros"!
6    answer = input("Take it or double it and send it to the next person?")
7    if answer.lower() == "take it":  # This choice does not end the game!
8        print("You took the money!")
9        continue
10   elif answer.lower() == "double it":
11       value *= 2
12   else:
13       print("Invalid input. Please respond 'take it' or 'double it'.")
```

Answer:

First, the `msg` in line 2 is defined once and never redefined. This is why the code always offer one dollar. We need to redefine the variable `msg` inside the loop, i.e. move line 2 in between lines 4 and 5. *(+5 points)*

Second, the run never stops because of the `continue` in line 9. That command makes the code jump to the next step of the same loop. We can fix it by changing it with `break`, which actually stops the loop. *(+5 points)*

(Optional, fixed code)

```
1  value = 1
2
3  while True:
4    msg = f"I offer you {value} euros"
5    print(msg)  # This is always "I offer you 1 euros"!
6    answer = input("Take it or double it and send it to the next person?")
7    if answer.lower() == "take it":  # This choice does not end the game!
8      print("You took the money!")
9      break
10   elif answer.lower() == "double it":
11     value *= 2
12   else:
13     print("Invalid input. Please respond 'take it' or 'double it'.")
```

9. The code below is intended to count the frequency of each character in a string. However, it raises a "`KeyError: "b"`" on the last line. Identify and fix the issue(s).

```
1 def char_frequency(s: str) -> dict:
2     freq = {}
3     for char in s.lower():
4         freq[char] += 1
5     return freq
6
7 print(char_frequency("BadBunny"))
8 # Expected: {"b": 2, "a": 1, "d": 1, "u": 1, "n": 2, "y": 1}
9 # Output: KeyError: 'b'
```

Answer:

The problem is line 4, which is the same as doing `freq[char] = freq[char] + 1`. If the `char` is not defined in the dictionary first, it fails (which is what currently happens). *(+5 points)*

We should include an if-clause inside the for loop: check if `char` is already a key in the dictionary. If it is, add 1 as we already do. If not, create the key and initialize it to 1. *(+5 points)*

(Optional, fixed code)

```
1 def char_frequency(s: str) -> dict:
2   freq = {}
3   for char in s.lower():
4     if char in freq.keys()
5       freq[char] += 1
6     else:
7       freq[char] = 1
8   return freq
```

10. You are given two lists of student names. The first list contains the students who attended a class, and the second list contains the students who submitted an assignment. Due to a registration error, some student names appear more than once in the same list. Your task is to fill the following report:

    - The total number of students who attended the class.
    - The total number of students who submitted the assignment.
    - The names of students who did both (attended and submitted).
    - The names of students who attended but did not submit.

    Design the code to generate this report. In which kind of Python data structure would you store each of these pieces of information? Explain your choices.

    Answer:

    The input of this function will be two list of strings: one belonging to the students who attended, and the other one for the students who submitted. *(+2 points)*

    Let's start by defining the output of our function. As we need to return several information, the best approach would be to return a dictionary with four keys:

    (a) `total_attended`: An integer. The total number of students who attended the class.

    (b) `total_submitted`: An integer. The total number of students who submitted the assignment.

    (c) `list_both`: A list of strings. The names of students who did both (attended and submitted).

    (d) `list_nosubmit`: A list of strings. The names of students who attended but did not submit.

    As we are going to return a dictionary, our function should initialize an empty dictionary using {}, and later on add the keys, using the structure `dictionary[new_key] = value`. *(+3 points)*

    Comment: While printing each result is also a possibility, it is not a good practice. Always try to return your results, so that we can use them later if needed! Using `print()` instead of a `return` statement is worth *+2* instead of *+3 points*.

    In order to avoid name duplications, the function first transforms the lists into sets, using the `set()` function. *(+5 points)*

    Next, we can count the number of students who attended and who submitted by simply using `len()` on their respective sets. We store these two numbers into the dictionary, as we mentioned before. *(+5 points)*

    Finally, to list the names of students who both attended and submitted, and the ones who only submitted, we will use set operations. To list both attend and submission, we compute the intersection between both sets. To list only attendance, we do the difference of attendance minus submission. In summary, we apply the relevant set operation, which returns a new set, then we turn that set back into a list using `list()`, and finally we store that into our dictionary. *(+5 points)*

Continue exercise 10

(Optional, code)

```python
def exercise10(ls_attend: list[str], ls_submission: list[str]) -> dict:
    # Initialize the output dictionary
    out_dict = {}

    # Turn list into sets
    set_attend = set(ls_attend)
    set_submission = set(ls_submission)
    # Count students that did each task
    out_dict["total_attended"] = len(set_attend)
    out_dict["total_submitted"] = len(set_submitted)

    # Use set operations for the other two ouputs
    set_both = set_attend.intersection(set_submission)
    set_nosub = set_attend.difference(set_submission)
    # Store them as list
    out_dict["list_both"] = list(set_both)
    out_dict["list_nosubmit"] = list(set_nosub)

    return out_dict
```

11. You are given a tuple of reports, each report consisting of a string of single digits separated by spaces, for instance:

```
reports = (
      "1 2 3",  # Valid
      "1 3 2",  # Invalid (not increasing)
      "1 4 4",  # Invalid (not strictly increasing)
      "1 2 3 4 5 6 7 8 9 10"   # Valid (any length)
  )
```

A report is considered valid if all digits are strictly increasing from start to end.

How would you write a function that, given **any** tuple of reports of **any size**, returns an integer: the number of valid reports?

Answer:

As we need to check several reports inside a tuple, the best approach will be to use a for loop. We will look one report at a time inside the loop, and apply the same rules to each of them. `for report in reports`. *(+5 points)*

Because we want to count the number of valid reports, before the loop starts we will initialize a count variable to 0. `count = 0`. Then, inside the loop, every valid report will increase that count by 1, using `count += 1`. At the end of the function, after the loop finishes, we return the resulting count. *(+5 points)*

Now we move inside the for loop. We have a single report, which is a string. Because we want to look at its individual numbers, it is preferable that we turn that string into a list of numbers. For instance, the report `"1 2 3"` should be converted to `[1, 2, 3]`. We can do this by chaining two operations. We first use the method `ls_strings = report.split(" ")` that returns a list of strings, splitting the report by its spaces. Next, to turn every string into a number, we can use list comprehension and the `int()` function, `ls_nums = [int(s) for s in ls_strings]`. *(+5 points)*

Now that we have that list of numbers, we initialize another for loop. This is a nested loop, inside the reports loop. This second loop is `for idx in range(1, len(ls_nums))`. The goal of this loop is to compare every pair of adjacent numbers inside the report. Using an if clause, we determine if the numbers are strictly increasing with `ls_nums[idx-1] < ls_nums[idx]`. Before we start the loop, we initialize a variable `is_valid = True`. If the if-clause is false, then we convert `is_valid = False` and break the nested loop. After this nested loop, we check wheter `is_valid` remains `True`, and if so, we increase the count by 1. *(+5 points)*

Continue exercise 11

(Optional, code)

```python
def exercise11(reports: tuple[str]) -> int:
    # Initialize the count of valid reports
    count_valid = 0
    # Loop through the tuple, looking at each report
    for string in reports:
        # example of string: "1 2 3"
        # Turn the report string into a list of strings
        ls_strings = string.split(" ")  # example: ["1", "2", "3"]
        # Turn each string into a number, using list comprehension
        ls_int = [int(s) for s in ls_strings]  # ex: [1, 2, 3]
        # We will assume the report is valid at first
        is_valid = True
        # Now loop through each number inside the report
        # (skipping the first one, we explain why later)
        for idx in range(1, len(ls_int)):
            # Compare every number against the previous one
            # (this is why we have skipped the first number)
            if ls_int[idx] <= ls_int[idx-1]:
                # If the number is lower or equal than the previous,
                # the report is not valid
                is_valid = False
                # A single non-valid occurence is enough,
                # so we stop this loop
                break
        # If our report has "survived" the validity test,
        # we can increase our count of valid reports
        if is_valid:
            count_valid += 1
    return count_valid
```