Python Journey

Introductory Problem Set

April 13, 2025

Disclaimer

In this handout we assume that students:

- Have already been introduced to Python functions, conditionals, for and while loops.
- Program through scripts instead of using the console.

The objective of these exercises is to reach a simple and intuitive solution, which not always corresponds to the most efficient or complete. Besides, some of these problems have built-in solutions in Python, but for learning purposes, we encourage students to "reprogram" said problems.

Exercise 1 - Even Odd

a) Develop a function that receives an integer and calculates whether it is odd or even.

b) Develop a test function for the Even Odd problem.

Exercise 2 - Grade Calculator

The following grading system is commonly used in the United States of America:

Letter Grade	Number Grade
A	90-100
B	80-89
C	70-79
D	60-69
F	0-59

a) Develop a function that receives an integer and calculates the corresponding letter grade. **Remember** to verify if the number received is within the desired scale.

b) Develop a test function for the Grade Calculator problem.

c) Develop an interactive program for the Grade Calculator problem. The user should also be able to terminate the program, for instance with the word "end". **Remember** to verify if the input is valid.

Exercise 3 - List Summation

Let $a = a_0, ..., a_{n-1}$, then a is sequence with n elements, starting from index 0 to $n-1$. The sum of the elements in sequence a, can be expressed as:

$$\sum_{i=0}^{n-1} a_i = a_0 + ... + a_{n-1}$$

Note that sequences can be stored as lists in python.

a) Develop a function that receives a list of floats and calculates the sum of its elements.

b) Develop a test function for the List Summation problem.

c) **Challenge:** Repeat the previous subproblems, but only sum the even numbers in the list.

Exercise 4 - Vowel Count

a) Develop a function that calculates the number of vowels in a string. **Warning**: The comparison between strings is **case-sensitive**.

b) Develop a test function for the Vowel Count problem.

Exercise 5 - Multiplication Table

Multiplication tables are often used when teaching students multiplication for the first time, one example being:

Tabuada do 2
$2 * 1 = 2$
$2 * 2 = 4$
$2 * 3 = 6$
$2 * 4 = 8$
$2 * 5 = 10$
$2 * 6 = 12$
$2 * 7 = 14$
$2 * 8 = 16$
$2 * 9 = 18$
$2 * 10 = 20$

a) Develop an interactive program that prints the multiplication table for numbers selected by the user one at a time. The user should also be able to terminate the program, for instance with the word "end". **Remember** to verify if the input is valid.

Exercise 6 - Fibonacci

There are innumerous ways to calculate the n^{th} number in the Fibonacci sequence. However, in this exercise students should use the iterative method, which means starting from the first and second Fibonacci numbers until one reaches the n^{th} number, using a loop.

$$\text{Fibonacci sequence: } 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...$$

a) Develop a function that calculates the n^{th} Fibonacci number.

b) Develop a test function for the Fibonacci problem.

c) **Challenge:** Repeat the previous subproblems for the Tribonacci problem, which instead of summing the two previous number, sums the three previous numbers. The Tribonacci sequence is as follows: $0, 1, 1, 2, 4, 7, 13, 24, 44, 81, ...$

Exercise 7 - Factorial

Similar to the Fibonacci sequence, there are various ways to calculate the factorial of a number, and once more, the objective will be to use its iterative version. However, in this case, one may iterate over the loop forward or backwards.

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$

a) Develop a function that calculates the factorial of a number n by looping forward, i.e., from 1 to n.

b) Develop a function that calculates the factorial of a number n by looping backwards, i.e., from n to 1.

c) Develop a test function for the Factorial problem.

Exercise 8 - Maximum Element

a) Develop a function that calculates the maximum element of a list of floats.
Suggestion: The result variable can be initialized as $float(inf)$

b) Develop a test function for the Maximum problem.

c) Repeat the previous subproblems for the minimum element.
Suggestion: The result variable can be initialized as $float(-inf)$

d) **Challenge:** Repeat the previous subproblems for the second-highest value of a list.

Exercise 9 - Prime Numbers

A prime number is any integer greater than one and only be divisible by itself and one. The first five prime numbers are: $2, 3, 5, 7, 11$.

a) Develop a function that calculates if a given integer is a prime number.

b) Develop a test function for the Prime Numbers problem.

Exercise 10 - Common Elements

a) Develop a function that calculates the number of common elements at the **same position** of two lists. **Warning:** the two lists may be of different sizes, as such, beware of indexes out of bound.

b) Develop a test function for the Common Elements problem.

Exercise 11 - Digit Sum

a) Develop a function that calculates the sum of all digits of an integer. **Suggestion:** Convert the integer to a string and only consider non-negative values.

b) Develop a test function for the Digit Sum problem.

Exercise 12 - Digital Root

The digital root of a number is the repeated sum of digits until the result only contains a single digit. For instance:

$$\text{Digit sum of } 9045 = 18\text{, because } 9 + 0 + 4 + 5 = 18$$

$$\text{Digital root of } 9045 = 9\text{, because } 9 + 0 + 4 + 5 = 18 \text{ and } 1 + 8 = 9$$

a) Develop a function that calculates the digital root of an integer. **Suggestion:** Use the function from the previous exercise until the stopping condition is met.

b) Develop a test function for the Digit Root problem.

Exercise 13 - Count Character

a) Develop a function that calculates the number of times a character occurs in a string. **Warning:** Remember that python does not support characters directly like other languages, as such, one must use a string and guarantee that it has length of one.

b) Develop a test function for the Count Character problem.

Exercise 14 - List Filter

a) Develop a function that filters the numbers from a list that are inferior to a certain threshold. **Suggestion:** The result should be a new list.

b) Develop a test function for the Inversion problem.

Exercise 15 - Inversion

a) Develop a function that inverts the order of the elements in a list. **Suggestion:** The result should be a new list.

b) Develop a test function for the Inversion problem.

Exercise 16 - Palindromes

A palindrome is a sequence of symbols that can be read forward and backwards in the same way. However, one may usually find that some liberties are taken, for instance with punctuation or formatting. For simplicity, we recommend only considering truly palindromic sequences of characters, i.e. the characters must exactly match in both ways, with the only exception being the letter cases, if the student wishes.

By our definition:

- "Stop pots" is truly palindromic, if one ignores letter cases;
- "Never odd or even" is not truly palindromic, since the 5th character from the start is "r" and from the back is a space.

a) Develop a function that determines whether a string is truly palindromic.

b) Develop a test function for the Palindromes problem.

Exercise 17 - Remove Element

a) Develop a function that removes a given element from a list. **Warning:** The element may be repeated multiple times in the list. **Suggestion:** The result should be in a new list.

b) Develop a test function for the Common Elements problem.

Exercise 18 - Remove Duplicates

a) Develop a function that removes the elements that are duplicated in a list. **Suggestion:** The result should be a new list.

b) Develop a test function for the Remove Duplicates problem.

Exercise 19 - Linear Search

Linear Search is the most intuitive and simple searching algorithm. It consists in going from position to position in a list and checking whether the element stored there is the desired element. The algorithm returns a boolean to indicate whether it was found.

a) Develop a function that implements linear search.

b) Develop a test function for the Linear Search problem.

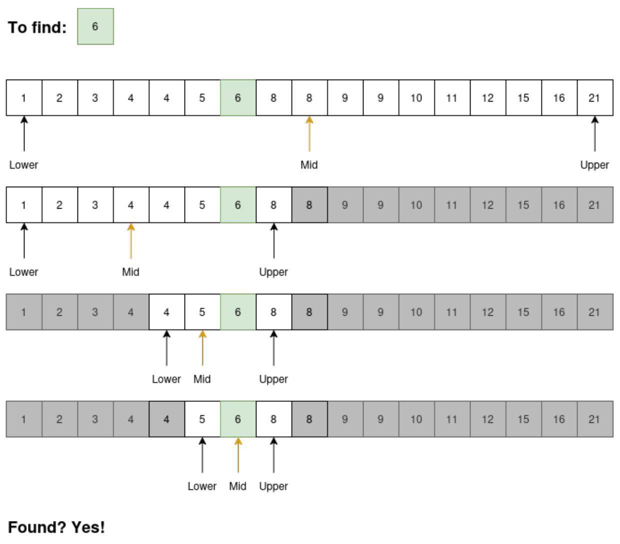
Exercise 20 - Sorted

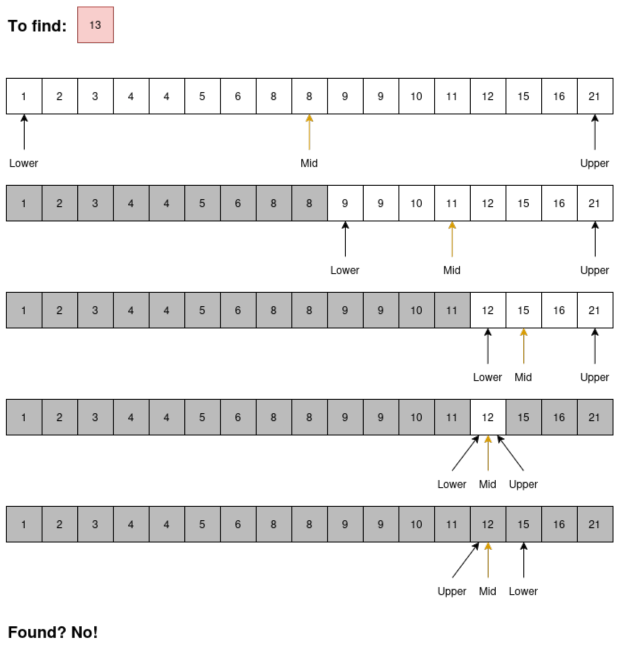
a) Develop a function to check if a list is sorted **ascendingly**.

b) Develop a test function for the Sorted problem.

Exercise 21 - Binary Search

Imagine that we have a sorted list, and we are looking for a specific value, would linear search be efficient? Possibly not, we might be looking for large value present at the end of a very large list. A common and efficient solution used in various fields of Engineering is to divide the domain in half, and if possible exclude one of the two halves. Binary Search does exactly that, since the list must be sorted beforehand, by checking the value at the middle point, we can restrict the domain in half in case the middle point is not the value we are looking for. This process can be repeated while the domain is valid. For clarity, see the following Images:





a) Complete the following function:

```
def bin_search(l:list[int], n: int) -> bool:
  assert is_sorted(l)

  lower = #TODO
  upper = #TODO

  while lower <= upper:
    mid = #TODO
    if l[mid] == n:
      return #TODO
    elif l[mid] > n:
      upper = mid - 1
    else:
      lower = mid + 1

  return #TODO
```

b) Develop a test function for the Binary Search problem.

Exercise 22 - Matrix Sum

Let A and B, be two $n * n$ matrices, then:

$$A + B = \begin{bmatrix} a_{11} & ... & a_{1n} \\ ... & ... & ... \\ a_{n1} & ... & a_{nn} \end{bmatrix} + \begin{bmatrix} b_{11} & ... & b_{1n} \\ ... & ... & ... \\ b_{n1} & ... & b_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} + b_{11} & ... & a_{1n} + b_{1n} \\ ... & ... & ... \\ a_{n1} + b_{n1} & ... & a_{nn} + b_{nn} \end{bmatrix}$$

For instance:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 3 \\ 4 & 6 & 6 \\ 7 & 8 & 10 \end{bmatrix}$$

a) Develop a function to check if the matrix has the same number of lines and columns.

b) Develop a test function for the previous subproblem

c) Develop a function that calculates the sum of two matrices. **Suggestions:** Guarantee that both matrices are $n * n$ and the result should be a new matrix.

d) Develop a test function for the Matrix Sum problem.

Exercise 23 - Matrix Transposition

Let A be $n * m$ matrix, and if:

$$A = \begin{bmatrix} a_{11} & ... & a_{1m} \\ ... & ... & ... \\ a_{n1} & ... & a_{nm} \end{bmatrix}, \text{ then } A^T = \begin{bmatrix} a_{11} & ... & a_{n1} \\ ... & ... & ... \\ a_{1m} & ... & a_{nn} \end{bmatrix}$$

For instance, if:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}, \text{ then } A^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix},$$

a) Develop a function to check if the list of lists has the same number of elements in every sub-list. In other words, if it is indeed an $n * m$ matrix, and every column has exactly m elements.

b) Develop a test function for the previous subproblem

c) Develop a function that calculates the transpose of a matrix. **Suggestions:** Guarantee that it is a valid matrix and the result should be a new matrix.

d) Develop a test function for the Matrix Transposition problem.

Exercise 24 - Interactive Average

a) Complete the following program, which calculates and prints the average of all numbers introduced by the user beforehand:

```
def interactive_avg():
  is_on = #TODO
  #TODO
  #TODO
  while is_on:
    val = input("Write a number: ")
    if val == "end":
      is_on = #TODO
    else:
      try:
        #TODO
        #TODO
        print("Average: ", """TODO""")
      except ValueError:
        print("Not a number!")

interactive_avg()
```

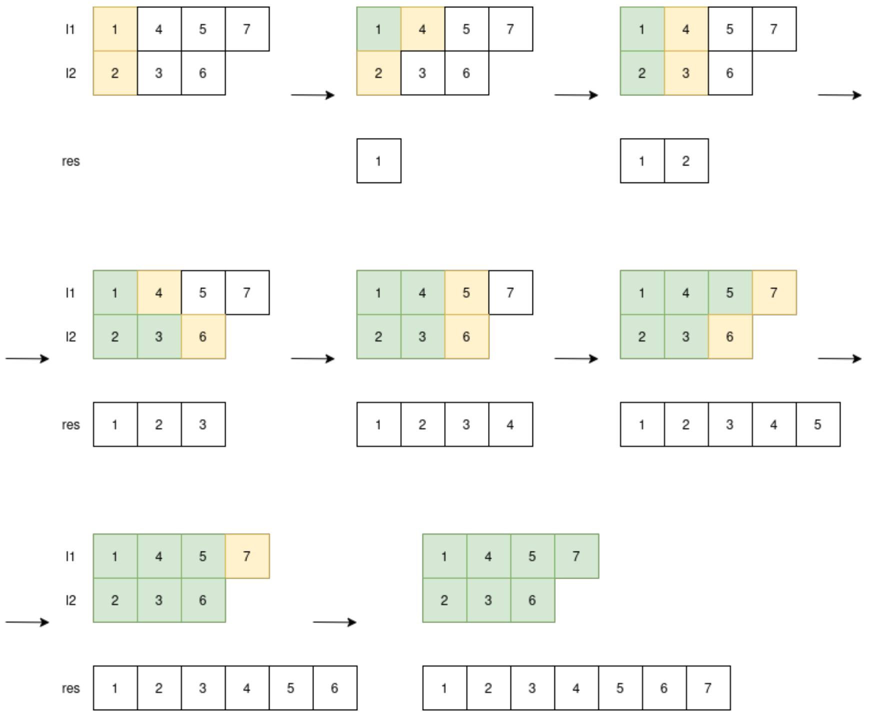
Exercise 25 - Sorted Concatenation

Concatenation commonly refers to the operation of joining two lists, usually the second list is placed at the end of the first. In Python this can be achieved with the 'extend' operation.

```
l1 = [1, 4, 5, 7]
l2 = [2, 3, 6]
l1.extend(l2)
# l1 = [1, 4, 5, 7, 2, 3, 6]

l3 = [1, 2, 3, 4]
l4 = [5, 6, 7]
l3.extend(l4)
# l4 = [1, 2, 3, 4, 5, 6, 7]
```

However, by naively concatenating two sorted lists the resulting list may lose that property (unless every element of $l1$ is lesser or equal that every element of $l2$). One possible solution would be to naively concatenate the two list then sort the result, which is not an interesting exercise, since Python offers a built-in function for sorting lists. Instead, we propose to build a new list (unlike 'extend' which alters the first list) and insert the element in order, for instance, for $l1$ and $l2$:



Suggestion: To achieve this, use a while loop that iterates over both lists at the same time, but only advance one element of one of the lists at a time.

a) Develop a function that performs the sorted concatenation of two lists.

b) Develop a test function for the Sorted Concatenation problem.

Exercise 26 - Bank Account

Consider an interactive program that simulates a simple ATM to access a bank account, and contains the following operations:

Button	Operation
1	Check balance
2	Deposit
3	Withdraw
4	Help
5	Quit

No need to consider any kind of security or account management aspects, or other operations (Imagine that you have your own ATM at home, which only knows (one of) your account(s))

a) Develop a function for the help operation. It should print the necessary information about the operations available.

b) Develop a function to print the balance.

c) Complete the following Deposit function:

```python
def deposit(currBalance):
  amount = None
  while amount == None:
    try:
      amount = float(input("Deposit amount: "))
      if """TODO""":
        print("Not a positive amount! Operation Cancelled")
        #TODO
      else:
        #TODO
    except:
      print("Not a number!")
```

d) Complete the following Withdraw function:

```python
def withdraw(currBalance):
  amount = None
  while amount == None:
    try:
      amount = float(input("Withdraw amount: "))
      if """TODO""":
        print("Not a positive amount! Operation Cancelled")
        #TODO
      elif """TODO""":
        print("Insufficient amount! Operation Cancelled")
        #TODO
      else:
        #TODO
    except:
      print("Not a number!")
```

e) Develop the main function of the ATM. **Suggestions:**

- It should follow the interactive program pattern (indefinite while loop that can be terminated by the user's input)
- Despite the previous buttons being represented as numbers, we are not interested in their numerical value, as such, it suffices using them as a string (In other words, no need to use the try/except shown before to avoid trying to convert things that may not be numbers into integers or floats)
- In console-based programming, where we have no other visual Elements other than text, it is good practice to display the help information at the start of the interactive program without the user requesting
- Do not forget to print a message if the user inputs something that does not appear in the table.
- Use the previous functions.