1.10   Problems

Problem 1.1 🔖BA   Write a program with the following requirements:

a. It defines variables for the following quantities:

$$x = 5.2 + j3.4, \quad y = -17, \text{ and } \quad z = 0.02,$$

where j is the imaginary number $\sqrt{-1}$.

b. It computes and prints the following quantities:

$$x + y, \quad xyz, \text{ and } \quad 4x^3 - 8xy + 6y^2.$$

c. It further computes and prints the following quantities:

$$|x|, \quad \overline{xy}, \text{ and } \quad \Re(x),$$

where $|\cdot|$ is the absolute value, $\overline{\cdot}$ is the complex conjugate, and $\Re(\cdot)$ is the real part.

Problem 1.2 🔖SB   Write a program with the following requirements:

a. It defines a variable for a list with the following elements:

```
4, -12, 6, -14, 8, -16
```

b. It prints the first and last elements of the list
c. Using list slicing, it prints the first three elements of the list
d. Using list slicing, it prints the last three elements of the list
e. Using list slicing, it prints every other element, starting with the first element
f. It computes and prints the length of the list (consider using the built-in function `len()`)
g. It computes and prints the sum of the list elements (consider using the built-in function `sum()`)

Problem 1.3 🔖I2   Write a program with the following requirements:

a. It defines a variable for a list with the following elements:

```
32, 41, 58, 34, 24, 53, 46, 41
```

b. It computes and prints the mean of the list items (consider using the built-in `sum()` and `len()` functions)
c. It finds and prints the maximum and minimum values in the list (consider using the built-in `max()` and `min()` functions)

d. It finds and prints the indices of the maximum and minimum values in the list (consider using the `index()` method)
e. It sorts and prints the sorted list (minimum to maximum; consider using the `sort()` method)

Problem 1.4 🔖VZ   Write a program with the following requirements:

a. It defines a function `which_number()` that takes a single argument and, if it is an `int`, `float`, or `complex` object, returns the strings `"int"`, `"float"`, or `"complex"`. If the argument is not a number, it returns `None`.
b. It tests the function and prints its return value on the following inputs:
   i. `42`
   ii. `3.92`
   iii. `complex(2, -3)`
   iv. `"3.92"`
   v. `[2, 0]`

Problem 1.5 🔖X4   Write a function `capital_only(l)` with the following requirements:

a. It accepts as input a list `l`
b. It checks that all elements are strings; it raises an exception, otherwise, with

```
raise ValueError(
    "All elements must be strings"
)
```

c. It returns a list (not the same list[2]) with only the strings that begin with a capital letter
d. It returns the proper output for the following inputs (demonstrate this in the program):
   i.   `["Foo", "Bar", "Baz"]`
   ii. `["Foo", "bar", "Baz"]`
   iii. `["Foo", 0, 1, "Bar", 2]`

Problem 1.6 🔖X0   Write a program with the following requirements:

a. It defines a function `float_list()` that takes a single `list` argument and returns a new list with all elements converted to `float`s
b. If the input is not a `list`, it returns an empty list
c. If an element is an `int`, it should be converted to a `float`
d. If an element is a `string`, the program should attempt to convert it to a `float`

---

2. Because a list is mutable, we must take care not to mutate a list inside a function (except in rare cases when this behavior is desired).

      i. For strings like `"3.24"`, the `float()` function will work
     ii. For strings like `"foo"`, the `float()` function will throw a **`ValueError`**; consider using **`try`** and **`except`** statements

e. If an element cannot be converted to a `float`, it should be left out of the returned list

f. If an element is `complex`, it should remain so

g. Test and print the returned list for the following inputs:

      i. `[1.1, 0.2, 4.2, -30.2]`
     ii. `[3, 42, -32, 0, 3]`
    iii. `[1-3j, 2, 0.3]`
    iv. `["1.2", "8", "-3.9"]`
     v. `["0.4", "dog", `**`None`**`, 8]`
    vi. `3.4`

2 The Structure, Style, and Design of Programs

With the development environment and basic elements of a Python program described in chapter 1, we can write a great many interesting programs. In this chapter, we consider how these programs should be structured and styled.

2.1 Python Interpreters and Interactive Sessions

When we execute (i.e., run) a Python program, an **interpreter** translates the program code into an efficient intermediate representation and carries out the corresponding instructions, which are ultimately represented in the lowest-level computer language called **machine code**. Instructions in machine code can be given directly to the processor and thereby executed.

An interpreter is a program that translates another program line-by-line. There is another way of translating a program (written in a programming language) to machine code—compiling. A compiler takes the entire program at once and translates it into a highly optimized machine code program, ready for execution. An interpreter cannot optimize a program as much as can a compiler, but there are advantages to using an interpreter, including that programs can be run interactively.

The official Python interpreter CPython was installed to our development environment in section 1.2. The basic way to run a Python program `hello.py` is in a terminal window with the command

```
python hello.py
```

Here the interpreter program `python` is called to interpret `hello.py` and the results are printed to the terminal. Programs written in a file like `hello.py` are called **scripts**.

Another way to run a Python script is within an **interactive session** (i.e., interactive shell, read-evaluate-print loop (REPL), or kernel) that runs lines of code as they are entered by the programmer. There are multiple programs that provide interactive Python sessions, including the standard one provided by the CPython distribution and invoked in a termal window with the command `python` (without