Solution 1.3 I2 The following program meets the requirements:

```
# a. Define list l
l = [32, 41, 58, 34, 24, 53, 46, 41]

# b. Mean
m = sum(l)/len(l)
print(f"Mean: {m}")

# c. Max and min
max_ = max(l)
min_ = min(l)
print(f"Max: {max_}; Min: {min_}")

# d. Indices of max and min
# Note: If there is duplication of max or min, the first index is found
print(f"Max Index (first): {l.index(max_)}")
print(f"Min Index (first): {l.index(min_)}")

# e. Sort
l.sort() # Mutates l itself (returns None)
print(f"Sorted: {l}")
```

This program prints the following to the console:

```
Mean: 41.125
Max: 58; Min: 24
Max Index (first): 2
Min Index (first): 4
Sorted: [24, 32, 34, 41, 41, 46, 53, 58]
```

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]`

Solution 1.4 VZ The following program meets the requirements:

```python
def which_number(x):
    """Identifies which type of number the argument is.

    Args:
        x: A number

    Returns:
        A string "int", "float", or "complex", or None.
    """
    t = type(x)
    if t == int:
        return "int"
    elif t == float:
        return "float"
    elif t == complex:
        return "complex"
    else:
        return None

# Test which_number()
test_args = [42, 3.92, complex(2, -3), "3.92", [2, 0]]
for arg in test_args:
    r = which_number(arg)
    # Print with repr() so that strings are displayed quoted
    print(f"which_number({repr(arg)}) => {repr(r)}")
```

This program prints the following to the console:

```
which_number(42) => 'int'
which_number(3.92) => 'float'
which_number((2-3j)) => 'complex'
which_number('3.92') => None
which_number([2, 0]) => None
```

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

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

c. It returns a list (not the same list[1]) 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]`

Solution 1.5 X4 The following program meets the requirements:

1. 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).

```python
def capital_only(l):
    """Return a list with only the strings from l that begin with
    capital letters.

    Args:
        l: A list of strings

    Returns:
        A new list of strings sans non-capitalized elements of l

    Raises:
        ValueError: If not all elements of l are strings
    """
    r = [] # Initialize return list
    for li in l:
        if type(li) == str:
            letter_start = li[0].isalpha() # First character is letter
            capitalized = li[0].capitalize() == li[0] # First is cap
            if letter_start and capitalized:
                    r.append(li)
        else:
            raise ValueError("All elements must be strings")
    return r

# Test capital_only()
test_args = [
    ["Foo", "Bar", "Baz"],
    ["Foo", "bar", "Baz"],
    ["Foo", 0, 1, "Bar", 2],
]
for arg in test_args:
    try:
        r = capital_only(arg)
        print(f"capital_only({repr(arg)}) => {repr(r)}")
    except ValueError as e: # Handle ValueError exception
        print(f"capital_only({repr(arg)}) => {type(e)}: {e}")
```

This program prints the following to the console:

```
capital_only(['Foo', 'Bar', 'Baz']) => ['Foo', 'Bar', 'Baz']
capital_only(['Foo', 'bar', 'Baz']) => ['Foo', 'Baz']
capital_only(['Foo', 0, 1, 'Bar', 2]) => <class 'ValueError'>: All
↪  elements must be strings
```

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 `str`ing, the program should attempt to convert it to a `float`
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`

Solution 1.6 X0 The following program meets the requirements:

```python
def float_list(l):
    """Return a list with all elements converted to floats.

    Args:
        l: A list

    Returns:
        New list with elements of l that could be converted to floats
    """
    if type(l) != list:
        return []
    r = [] # Initialize return list
    for li in l:
        if type(li) == float or type(li) == complex:
            r.append(li)
        else:
            try: # converting to a float
                li_float = float(li) # If this throws exception then
                r.append(li_float)   # ... this will not execute
            except:
                continue
    return r

# Test float_list()
test_args = [
    [1.1, 0.2, 4.2, -30.2],
    [3, 42, -32, 0, 3],
    [1-3j, 2, 0.3],
    ["1.2", "8", "-3.9"],
    ["0.4", "dog", None, 8],
    3.4,
]
for arg in test_args:
    r = float_list(arg)
    print(f"float_list({repr(arg)}) => {repr(r)}")
```

This program prints the following to the console:

```
float_list([1.1, 0.2, 4.2, -30.2]) => [1.1, 0.2, 4.2, -30.2]
float_list([3, 42, -32, 0, 3]) => [3.0, 42.0, -32.0, 0.0, 3.0]
float_list([(1-3j), 2, 0.3]) => [(1-3j), 2.0, 0.3]
float_list(['1.2', '8', '-3.9']) => [1.2, 8.0, -3.9]
float_list(['0.4', 'dog', None, 8]) => [0.4, 8.0]
float_list(3.4) => []
```