R2

1. Given the following code fragment what is the result of the execution? Explain your answer **[2p]**.

```
a = 1
b = [str(a), 1]
c = {a: a, str(b[1]): b }
print(type(a) == type(c[1]))
```

```
a = ['1']
print(type(a) == type(c[1]))
print(type(b[0]) == type(a))
print(str(a) in c)
```

2. A sparse data structure is one where we presume most of the elements have a common value (e.g. 0). Write the **SparseMatrix** class, which implements a sparse matrix data structure, so that the code below works as explained in the comments. Elements not explicitly set have the default value 0. The matrix size is given in the constructor. Do not represent the 0 values in memory! **[2p]**

```
# Initialize a 3x3 sparse matrix
m1 = SparseMatrix(3,3)
# Value at [1,1] is 2
m1.set(1,1,2)
# Value at [2,2] is 4
m1.set(2,2,4)
# Prints
# 0 0 0
# 0 2 0
# 0 0 4
print(m1)
# Prints '<class 'ValueError'>'
try:
    m1.set(3,3,99)
except Exception as e:
    print(type(e))
# Update value at [1,1] with 2 + 1
m1.set(1,1,m1.get(1,1)+1)
# Prints
# 0 0 0
# 0 3 0
# 0 0 4
```

3. Analyze the time complexity of the following function **[1p]**.

```
def f(n):
    s = 0
    for i in range(1, n * n + 1):
        j = 1
        while j < n:
            s = s + j
            j = j * 2
    return s
```

*Write the specification, python code and test cases for a **recursive function** that calculates the sum of even numbers found on even positions in a list, using the divide and conquer method. For the input [2, 2, 4, 5, 6, 4, 13, 4, 10], the result is 22* **[2p]**.

For the following problem statement, indicate the most appropriate programming method (between backtracking, divide and conquer, greedy or dynamic programming) that can be used for solving it. Justify the method's applicability and analyze the problem solving according to the particularity of the selected programming technique (implementation not required) **[2p]**.

Determine all the permutations for a given natural number **n**, where elements to the left of the maximum value are increasing and those to the right are decreasing. **e.g.** for n = 3, we get 0,1,2,3 | 0,1,3,2 | 0,2,1,3 | 0,3,2,1 | 1,2,3,0 | 1,3,2,0 | 2,3,1,0 | 3,2,1,0

... implements a sparse matrix data structure, so that the code below ... have a common value (e.g. 0). ... explained in the comments. Elements not explicitly set have the default value 0. The matrix size is given in the constructor. Do not represent the 0 values in memory! **[2p]**

```
# Initialize a 3x3 sparse matrix
ml = SparseMatrix(3,3)
# Value at [1,1] is 2
ml.set(1,1,2)
# Value at [2,2] is 4
ml.set(2,2,4)
# Prints
# 0 0 0
# 0 2 0
# 0 0 4
print(ml)
# Prints '<class 'ValueError'>'
try:
    ml.set(3,3,99)
except Exception as e:
    print(type(e))
# Update value at [1,1] with 2 + 1
ml.set(1,1,ml.get(1,1)+1)
# Prints
# 0 0 0
# 0 3 0
# 0 0 4
```

... analyze the time complexity of the following function **[1p]**.

```
def f(n):
    s = 0
    for i in range(1, n * n + 1):
        j = 1
        while j < n:
            s = s + j
            j = j * 2
    return s
```

4. Write the specification, python code and test cases for a ***recursive function*** that calculates the sum of even numbers found on even positions in a list, using the divide and conquer method. For the input **[2, 2, 4, 5, 6, 4, 13, 4, 10]**, the result is 22 **[2p]**.
5. For the following problem statement, indicate the most appropriate programming method (between backtracking, divide and conquer, greedy or dynamic programming) that can be used for solving it. Justify the method's applicability and analyze the problem solving according to the particularity of the selected programming technique (implementation not required) **[2p]**.

Determine all the permutations for a given natural number **n**, where elements to the left of the maximum value are increasing and those to its right are decreasing. **e.g.** for n = 3, we get 0,1,2,3 | 0,1,3,2 | 0,2,3,1 | 0,3,2,1 | 1,2,3,0 | 1,3,2,0 | 2,3,1,0 | 3,2,1,0

Hammurabi

You are the mighty Hammurabi, elected into office for a five year term as ruler of Sumeria. Each year, you have to decide your capital city's land management policy, make sure that your people do not starve, and plant next year's crops. The currency is represented by units of grain. The game lasts for 5 turns. Each turn is 1 year.

1. Each year, your trusted advisor reports the current situation, in the format at the bottom of the page. Use the data from Year 1 below when starting a game **[1p]**:
2. Each year, you decide how many acres of land to buy or sell, how many units of grain to feed your population with, and how many acres of land are planted with grain **[0.5p]**. You cannot buy more land than you have grain for, you cannot sell more land than you have, you cannot feed people with grain you do not have, you cannot plant more acres than you have, you cannot plant grain that you do not have **[1p]**.

The next steps happen without your intervention:

3. Land is bought or sold at the given price, and the stock of grain is updated. Each year, land prices vary between 15 and 25 grain per acre (random) **[1p]**.
4. Grain used to feed the population is subtracted from the stock. Each person needs 20 units of grain to survive. People who do not receive this amount starve **[1p]**. If half your population starves during the year, the game is over **[1p]**. If no one starved, between 0 and 10 people (random) come to the city **[1p]**.
5. Grain is harvested. Each person can ~~harvest~~ plant at most 10 acres. Each year, the harvest is random between 1 and 6 units of grain per acre. **[1p]**
6. Each year, there is a 20% chance of a rat infestation. Rats eat at most 10% of the stored grain. **[1p]**

When the five years are up:

7. The game ends. If you have over 100 population and over 1000 acres, display a congratulation message. Otherwise display a loss message **[0.5p]**

Non-functional requirements:

- Implement a layered architecture solution.
- The program must not crash, regardless of user input.
- Provide specification and tests for functions processing the year.
- The exam cannot be passed without working functionalities!

In year 1, 0 people starved. 0 people came to the city. City population is 100 City owns 1000 acres of land. Harvest was 3 units per acre. Rats ate 200 units. Land price is 20 units per acre. Grain stocks are 2800 units. **Acres to buy/sell(+/-)-> 0** **Units to feed the population-> 2000** **Acres to plant-> 800**	In year 2, 0 people starved. 5 people came to the city. City population is 105 City owns 1000 acres of land. Harvest was 2 units per acre. Rats ate 0 units. Land price is 22 units per acre. Grain stocks are 1600 units. **Acres to buy/sell(+/-)-> -100** **Units to feed the population-> 2100** **Acres to plant-> 900**	In year 3, 0 people starved. 5 people came to the city. City population is 110 City owns 900 acres of land. Harvest was 1 units per acre. Rats ate 0 units. Land price is 16 units per acre. Grain stocks are 1700 units. **Acres to buy/sell(+/-)-> -100** **Units to feed the population-> 2000** **Acres to plant-> 800**
In year 4, 10 people starved. 0 people came to the city. City population is 100 City owns 800 acres of land. Harvest was 1 units per acre. Rats ate 0 units. Land price is 16 units per acre. Grain stocks are 1300 units. **Acres to buy/sell(+/-)-> -100** **Units to feed the population-> 2000** **Acres to plant-> 700**	In year 5, 0 people starved. 9 people came to the city. City population is 109 City owns 700 acres of land. Harvest was 3 units per acre. Rats ate 300 units. Land price is 24 units per acre. **GAME OVER. You did not do well.**	

Default 1p.

R2

1. Given the following code fragment what is the result of the execution **(1p)**:

```
def f(a, b, c):
    a = a+1
    b.append(3)
    c = c+[3]

a = 7
b = [1, 2]
c = [1, 2]
f(a, b, c)
print a,b,c
```

a) print: 7 [1,2] [1,2]
b) print: 8 [1,2,3] [1,2,3]
c) print: 7 [1,2,3] [1,2]
d) error on line: c = c+[3]

2. Please specify and test the following function: **(2p)**

```
def f(l):
    if (l==None):
        raise ValueError()
    for e in l:
        if e%2==1:
            return True
    return False
```

3. Asymptotic analysis of the time complexity (best case, average case, worst case). Please also indicate the extra-space complexity. **(2p)**

```
def f(n):
    """
      n - integer number
    """
    s = 0
    m = n
    while (m!=0):
        s=s+m/10
        m = m/10
    return s
```

```
def a(n):
    s = 0
    for i in range(1,n+1):
        m = 2*n+1
        s = s+ f(m)
    return s
```

4. Let us consider a list $a_1, a_2, \ldots, a_n$ of integer numbers. Using the „Divide et Impera" programming method, write ~~write~~, specify and test a function to compute the number of even elements from the list. **(2p)**

5. For the following problem, please indicate the most APPROPRIATE programming method (Backtracking, Divide et Impera, Greedy, Dynamic Programming) that can be used for solving it. Please justify the method's applicability and analyse the problem solving according to the particularity of the selected programming technique (without implementation) **(2p)**

Give all the possibilities to decompose a given natural number **n** as a sum of prime numbers. (**Example**. For **n=15** the solutions are **3, 5, 7** and **2, 13**).