Part 2 - Examples:

OO-Programming

The questions in Part 2 tend to be easy to medium.
There will also be more difficult questions on the exam.

Given is the following **class Account**:

```
class Account:
    def __init__(self, owner, balance=0):
        self.owner = owner
        self.balance = balance       # for example in Dollars!

    def __str__(self):
        return f'Account owner:   {self.owner}\nAccount balance: ${self.balance}'

    def deposit(self, dep_amt):
        self.balance += dep_amt
        print( 'Deposit Accepted: $', dep_amt )

    def withdraw(self, wd_amt):
        if self.balance >= wd_amt:
            self.balance -= wd_amt
            print( 'Withdrawal Accepted: $', wd_amt )
        else:
            print('Funds Unavailable!')
```

To do:

Read the 7 commands below in the function **main()** and program them yourself in the textbox below.
Note: First, have a look at the **required result-output** below! You will find some hints there!

```
def main():
    # 1. Create an account object with the name 'acc01'
    #    for yourself (this means with your 'name' and 100 Dollar)
    # 2. Print the object 'acc01' on the console
    # 3. Print the account owner attribute only
    # 4. Print the account balance attribute only
    # 5. Make a of deposit of 50$
    # 6. Make a withdrawal of 75$ Show (with print function)
    # 7. Make a withdrawal that exceeds the available balance

if __name__ == "__main__":
    main()
```

Your Solution: (There is always enough space in the exam!)

Here the **requried result-output**:

```
# Account owner:   Erwin
# Account balance: $100
# Erwin
# 100
# Deposit Accepted: $50
# Withdrawal Accepted: $75
# Funds Unavailable!
```

Question:

```
class Account:
    def __init__(self, owner, balance=0):
        self.owner = owner
        self.balance = balance  # for example in Dollars!

    def __str__(self):
        return f'Account owner: {self.owner}\nAccount balance: $

    def deposit(self, dep_amt):
        self.balance += dep_amt
        print('Deposit Accepted: $', dep_amt)

    def withdraw(self, wd_amt):
        if self.balance >= wd_amt:
            self.balance -= wd_amt
            print('Withdrawal Accepted: $', wd_amt)
        else:
            print('Funds Unavailable!')
```

Answer

```
def main():
    # 1. Create an account object with the name 'acc01' for your
    acc01 = Account('Erwin', 100)

    # 2. Print the object 'acc01' on the console
    print(acc01)

    # 3. Print the account owner attribute only
    print(acc01.owner)

    # 4. Print the account balance attribute only
    print(acc01.balance)
```

```
    # 5. Make a deposit of 50$
    acc01.deposit(50)

    # 6. Make a withdrawal of 75$ (show with print function)
    acc01.withdraw(75)

    # 7. Make a withdrawal that exceeds the available balance
    acc01.withdraw(500)

if __name__ == "__main__":
    main()
```

Rectangle Calculation

Create a function which calculates the **area**, the **circumference** and the **diagonal** of a rectangle with the parameters a = length, b = width.

The name of the function is: **RectValues**

```
#------------ here starts is the code --------------------------
import math

# the following missing function calculates the area, circumferences and the diagonal of a rectangle and
returns all 3 values at once with a list or tuple

# program the full (including the header) function RectValues(...) in the text box at the bottom
 def RectValues(...
    ...
    ...
    ...


# the next lines are only for testing purposes of your function and don't have to be copied to the text
box.

mylength = input("Length of the recangle: ")
mywidth = input("Width of the recangle: ")

myRectValues = RectValues(mylength,mywidth)

print(myRectValues)
print("Area: %8.2f, perimeter: %8.2f, diagonal: %8.2f" % (
myRectValues[0],myRectValues[1],myRectValues[2]) )
```

```
import math

# Define the function RectValues
def RectValues(a, b):
    area = a * b
    circumference = 2 * (a + b)
    diagonal = math.sqrt(a**2 + b**2)
    return area, circumference, diagonal

# The next lines are only for testing purposes of your function
myLength = float(input("Length of the rectangle: "))
myWidth = float(input("Width of the rectangle: "))

myRectValues = RectValues(myLength, myWidth)

print(myRectValues)
print("Area: %8.2f, Perimeter: %8.2f, Diagonal: %8.2f" % (
    myRectValues[0], myRectValues[1], myRectValues[2]))
```

Task 1:

Iterator & Iterable (8 Pt)

Frage 1 von 5 (8 Punkte)
Nicht beantwortet

Given is following class:

```
class DownSizeMutable:
    def __init__(self, start, downsize = 1):
        self.current = start
        self.downsize = downsize

    def __iter__(self):
        return self

   def __next__(self):
        if self.current <= 0:
            raise StopIteration
       else:
            self.current -= self.downsize
            return self.current + self.downsize
```

Tasks A:

Think about: *Are objects of the class* `DownSizeMutable` *'iterable'? Answer: YES!!!*

Program a function `isIterable(object) -> bool` that proves if the passed object is iterable.
Call the function **isIterable(...)** with the following short python script:

```
dsm = DownSizeMutable(70, 7)
print(isIterable(dsm))
```

Task B:

Demonstrate with a **for-loop** that the object `dsm` is iterable. Save the values of `DownSizeMutable(70, 7)` in a list with the name **myList**.
Print out the content of **myList**.
The printed result will be: `[70, 63, 56, 49, 42, 35, 28, 21, 14, 7]`

Task C:

The result of the task C is the same as in TaskB.
BUT: Solve the same task with a **while-loop**.

Important:

- Copy your solution for Task A, B and C into the text box below.
- The different parts of these 3 tasks (Task A, Task B and Task C) must be **clearly labeled** in the text box below.
  -> **In short:** It must be very clear, which code belongs to which task (use comments, titles, lables, etc.)

Task A:

```
def isIterable(obj):
    try:
        iter(obj)
        return True
    except TypeError:
        return False

# Testing the function
dsm = DownSizeMutable(70, 7)
print(isIterable(dsm))  # Output should be True
```

Task B: Use a For-Loop to Iterate

We need to demonstrate that the object `dsm` is iterable using a for-loop and save the values in a list `myList`.

```
# Initialize the object
dsm = DownSizeMutable(70, 7)

# Create an empty list to store the values
myList = []

# Use a for-loop to iterate over the object
for value in dsm:
    myList.append(value)

# Print the list
print(myList)  # Output should be [70, 63, 56, 49, 42, 35, 28, 2
```

Task C: Use a While-Loop to Iterate

```
# Initialize the object again since it's been consumed in Task B
dsm = DownSizeMutable(70, 7)
```

```
# Create an empty list to store the values
myList = []

# Use a while-loop to iterate over the object
while True:
    try:
        value = next(dsm)
        myList.append(value)
    except StopIteration:
        break

# Print the list
print(myList)  # Output should be [70, 63, 56, 49, 42, 35, 28, 2
```

TASK 2: Loop

Loop (8 pt)

Frage 2 von 5 (8 Punkte)
Nicht beantwortet

The given 2D list contains some empty (NA = no value is available) elements. Write a function called **fixMatrix(....)** that locates missing elements and fills them up by linear interpolation of the previous and next (adjacent = benachbart) elements in the same row. So, if the element E22 is missing for instance, it has to be filled up using the following formula:

```
E22 = (E21+E23)/2
```

The 2D list looks like:

```
[[E00, E01, E02, E03],
 [E10, E11, E12, E13],
 [E20, E21, E22, E23],
 [E30, E31, E32, E33]]
```

Tasks to solve:

Program a function **fixMatrix(...)** with the following rules for edge conditions:

- If there is no previous element (i.e. the missing element is the first of the row) use the last element of previous row. Example: `E10 = (E03+E11)/2`
- If there is no next element (i.e. the missing element is the last of the row) use the first element of next row. Example: `E23 = (E22+E30)/2`
- The first element (i.e. E00) and the last element (i.e. Enn) are never 'NA's.
- There are never multiple 'NA' next to each other (=nebeneinander).

Write the function as generic as possible and try it with the following two 2D lists:

```
missingEl1 = [[   1,'NA',   3,   4],
              ['NA',   6,'NA',   8],
              [   9,  10,  11,  12],
              [  13,'NA',  15,  16]]


missingEl2 = [[   1,'NA',   3,   4,   5],
              [  11,'NA',  13,  14,'NA'],
              [  21,  22,'NA',  24,  25],
              [  31,  32,  33,'NA',  35],
              ['NA',  42,  43,  44,  45],
              ['NA',  52,'NA',  54,  55]]
```

Some Code Hints:

```
def fixMatrix(matrix):
    rows = len(matrix)           # number of rows
    cols = len(matrix[0])        # number of cols
    ...
print(fixMatrix(missingEl1))
print(fixMatrix(missingEl2))
```

Important:

Copy your whole solution into the text box below.

```
def fixMatrix(matrix):
    rows = len(matrix)  # Number of rows
    cols = len(matrix[0])  # Number of columns
```

```
    for i in range(rows):
        for j in range(cols):
            if matrix[i][j] == 'NA':
                # Determine the previous element
                if j == 0:  # First element of the row
                    if i == 0:  # Special case for the very firs
                        continue
                    prev_value = matrix[i-1][-1]  # Last element
                else:
                    prev_value = matrix[i][j-1]

                # Determine the next element
                if j == cols - 1:  # Last element of the row
                    if i == rows - 1:  # Special case for the ve
                        continue
                    next_value = matrix[i+1][0]  # First element
                else:
                    next_value = matrix[i][j+1]

                # Calculate the missing value using linear inter
                matrix[i][j] = (prev_value + next_value) / 2

    return matrix

# Test with provided 2D lists
missingE1 = [
    [1, 'NA', 3, 4],
    ['NA', 6, 'NA', 8],
    [9, 10, 11, 12],
    [13, 'NA', 15, 16]
]

missingE2 = [
    [1, 'NA', 3, 4, 5],
    [11, 'NA', 13, 14, 'NA'],
    [21, 22, 'NA', 24, 25],
```

```
    [31, 32, 33, 'NA', 35],
    ['NA', 42, 43, 44, 45],
    ['NA', 52, 'NA', 54, 55]
]

print(fixMatrix(missingE1))
print(fixMatrix(missingE2))
```

TASK 3: File Write

Note:

This is an independent task. That means, the "Logger" class in this task has nothing (!!!) to do with any other task of the exam!

Tasks to solve:

Write a simple logger class (class name: "**Logger**") that creates an append-only file when a new log-object is initiated.
The logger comprises:

- a private entry number generator starting with 1 and increments with each new log entry
- one function '**add_entry()**' to append a new log entry to the file 'log.txt' starting with the entry number at the begining.
- each entry number is written with minimum 5 characters and enclosed in square brackets.

Two typical log entries look like:

```
[    1] First entry.
[    2] Second entry.
...
[  125] This is the 125th entry.
```

With this 'logger test script' you can test your Logger class:

```
logger = Logger('log.txt')
logger.add_entry('First entry.')
logger.add_entry('Second entry.')
logger.add_entry('The PDS MEP is running!')
```

Important:

- Copy your whole solution (with the 'logger test script') into the text box below.
- Copy also the content of your generated 'log.txt' File on the end of your solution.

```
class Logger:
    def __init__(self, file_name):
        self.file_name = file_name
        self.entry_count = 1  # Initialize the entry counter

    def add_entry(self, entry_text):
        # Format the entry number with a minimum width of 5 char
        entry_number = f"[{self.entry_count:5}]"
        # Create the log entry string
        log_entry = f"{entry_number} {entry_text}\n"

        # Open the file in append mode and write the log entry
        with open(self.file_name, 'a') as log_file:
            log_file.write(log_entry)

        # Increment the entry counter for the next entry
        self.entry_count += 1

# Test the Logger class
logger = Logger('log.txt')
logger.add_entry('First entry.')
logger.add_entry('Second entry.')
logger.add_entry('The PDS MEP is running!')
```

TASK 4: Inheritance

Inheritance (8 pt)

Frage 4 von 5 (8 Punkte)
Nicht beantwortet

Tasks to solve:

- Create the class **MyStr** which extends the built-in class **str** with a property **wordCount** that returns the word count of the string.

Note:

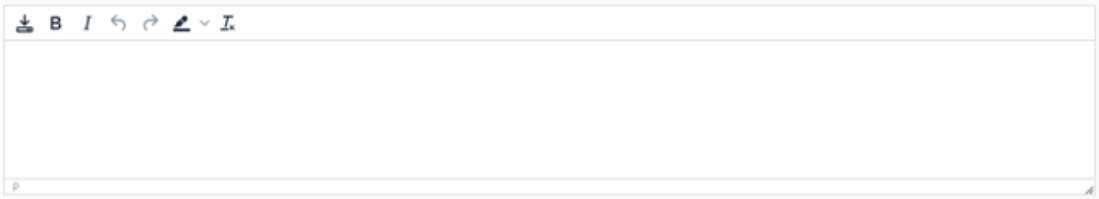
Words are separated by spaces. That means, numbers in text also count as words.

The following 'inheritance test script' should print the number 69 with your implementation of **MyStr**.

```
# 'inheritance test script':
txt = MyStr("Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.")
print(txt.wordCount)          # prints the number 69 to console (there are 69 words in the text!)
```

Important:

Copy your whole solution (with the class Mystr and the 'inheritance test script') into the text box below.



```
class MyStr(str):
    @property
    def wordCount(self):
        # Split the string by spaces to get a list of words
        words = self.split()
        # Return the length of the list, which is the number of
        return len(words)

# Inheritance test script
txt = MyStr("Lorem ipsum dolor sit amet, consectetur adipiscing
print(txt.wordCount)  # prints the number 69 to console (there a
```

Task 5: Lambda and list Comprehension

← Zurück

Lambda function & list comprehension (8 pt)

Frage 5 von 5 (8 Punkte)
Nicht beantwortet

Given is following code:

```
myList = [5.69, 9, 8.5, 7.36, 9.52, 7, 6.01, 6.68, 2.28, 9.78, 6, 9.33, 2.05, 3.40, 7.76, 6.1, 5.68]
def myfilter(mylist, lamFunc):
    return [lamFunc(x) for x in mylist]   # returns the list resulting from the list comprehension
```

Tasks A:

program a **lambda function** 'decimal_part' with

```
decimal_part = lambda x: ...# your task ! ..............
```

which is called in the following script:

```
result = myfilter(myList, decimal_part)
print(result)
```

The lambda function should return all decimal parts of `myList`.
The result of the `print(result)` should be something like:

```
[0.69, 0, 0.5, 0.36, 0.52, 0, 0.01, 0.68, 0.28, 0.78, 0, 0.33, 0.05, 0.4, 0.76, 0.1, 0.68]      # 9, 7 and 6
have NO decimal places (=Dezimalstellen)
```

Tasks B:

Change the **list comprehension** in the function `myfilter(mylist, lamFunc)` such that the resulting list for the given `myLlist` only contains decimal values >0.

The final list of `print(result)` should look as follow:

```
[0.69, 0.5, 0.36, 0.52, 0.01, 0.68, 0.28, 0.78, 0.33, 0.05, 0.4, 0.76, 0.1, 0.68]      # all 0 values removed
```

Here are some code snippets that might be of some help to you... ;-):
Check out the solutions and benefit!

```
print("5 != int(5):", 5 != int(5))
print("5.1 != int(5.1):", 5.1 != int(5.1))
print("int(5.1):", int(5.1))
print("int(0.33):", int(0.33))

print("5 % 3 = ", 5%3)
print("5.1 % 3 = ", 5.1%3)
print("round(5.1 % 3, 2): ", round(5.1 % 3, 2))
print("round(5.666 % 1, 2): ", round(5.666% 1, 2))
```

Important:

Copy your solution of task A and task B individually or (also possible) together in ONE solution into the text box below.

```
myList = [5.69, 9.85, 7.36, 9.52, 7.6, 6.01, 6.68, 2.28, 9.78, 6

# Task A: Lambda function to get the decimal part of a number
decimal_part = lambda x: x - int(x)
```

```
# Task B: Modified myfilter function to filter and keep only nor
def myfilter(mylist, lamFunc):
    return [lamFunc(x) for x in mylist if lamFunc(x) > 0]

# Apply the filter
result = myfilter(myList, decimal_part)

# Testing the solution with .2f formatting in the print statemer
formatted_result = [f"{num:.2f}" for num in result]
print(formatted_result)  # Output: ['0.69', '0.85', '0.36', '0.5
```