LAB 03

1. Write a function that calculates the surface area of a cylinder based on the radius and height entered as command line arguments.

Hint: the math python module provides a way to get the value of Pi (see Python documentation here https://docs.python.org/3/library/math.html).

2. Write a function x(n) for computing an element in the sequence $x_n=n^2+1$. Call the function for n=4 and print out the result to the console.
3. Write a function that evaluates the mathematical functions

$$f(x) = \cos(2x), f'(x) = -2\sin(2x), \text{ and } f''(x) = -4\cos(2x).$$

Return these three values. Write out the results of these values for x = Pi.

4. Python comes with a set of built-in functions a developer can use in her/his code. *print()* is an example built-in function. It outputs a python object into the console.

Check the Python documentation for a list of available built-in functions.

https://docs.python.org/3/library/functions.html

By clicking on one of these functions you can get the details of each of them including what the function does, what is its expected input and return value(s) if any.

Write a python code that makes use of some of these built-in functions.

5. Write a function called *accept_login(users, username, password)* with three parameters: users; a dictionary of username keys and password values, username a string for a login name and password a string for a password. The function should return True if the user exists and the password is correct and False otherwise. Here is the calling code, test your code with both good and bad passwords as well as non-existent login names:

```
users = {
   "user1" : "password1",
   "user2" : "password2",
   "user3" : "password3"}
if accept_login(users, "user", "password") :
   print("login successful!")
else:
      print("login failed...")
```

6. Byte converter

We want to combine the usage of tuples and loops to create a function, which converts bytes into bytes, kilobytes, megabytes or gigabytes and returns the correct unit.

In everyday life the metric system is used, meaning every conversion is a multiple of 10. Kilo for example has a conversion factor of 10^3 = 1000, so 1 kilometre is 1000m. In computing the binary system is used in most cases, which uses a base of 2 instead of 10. Thus, 2^10 = 1024 is used to convert between bytes and kilobytes, kilobytes and megabytes, and so on.

Hint: Check out the *math.pow(x, y)* function in the python documentation https://docs.python.org/3/library/math.html

- First, we need to set up the units as a tuple, so that the result can be printed with the units attached.

```
units = ("B", "KB", "MB", "GB")
```

- A short note on tuples: tuples are unchangeable. In case someone else uses your code, you do not want anybody to change the units of this method. The same goes for example for PI, PI is always roughly 3.1415926535, nobody should be able to change that.
- At the beginning we know the initial value in bytes, after one iteration we should know the size in kilobytes, etc. So we will have to keep track of the size in the current unit. To get higher accuracy, we also want to know the decimal number of bytes, so that 512 bytes would return “0.5 KB”.
- As a test you can use the following result. Entering 537395200 as bytes, should display:

```
537395200 B
524800 KB
512.5 MB
0.5 GB
```

7. Counting lines in files

Tutorial: Python command line arguments

Create a script *argumentsTest.py* as follows:

```
1.  #!/usr/bin/env python3
2.  import sys
3.
4.  print (‘List of arguments:', sys.argv)
5.  print ('Total number of arguments:', len(sys.argv), 'arguments.')
```

Now in python interactive mode, execute the script using the following command:

```
User123$ python3 argumentsTest.py arg1 arg2 arg3
```

You will obtain an output similar to the following:

```
List of arguments: ['argumentsTest.py', 'arg1', 'arg2', 'arg3']
Total number of arguments: 4 arguments.
```

Explanation:

In line 2, the imported **sys** module provides access to any command-line arguments via the **sys.argv**, where:

- sys.argv in line 4, represents is the list of arguments
- len(sys.argv) in line 5 represents the number of command-line arguments

Note: By default, the first argument is the script name.

Example: sys.argv[0] is the script name 'argumentsTest.py'

Task

Write a python code '`CountLines.py`'. The script will take as input (command line arguments) a list of file names and outputs the number of lines in each one of these files. Use text files. Here is an example of how to run your script:

```
python3 CountLines file1.txt file2.txt file3.txt
```

Write each file name, along with the number of lines in that file, to standard output.

Note: Exceptions handling should be implemented as well. If an error occurs while trying to read from one of the files, you should print an error message for that file, but you should still process all the remaining files.

8. Date, time and generators

Follow the tutorial on python generators available here:
https://www.programiz.com/python-programming/generator

Write a python program that makes use of a generator that produces a list of dates between two dates.
The start and end dates can be hard-coded in your program like this.

```
startDate  = date(2025, 4, 27)
endDate    = date(2025, 5, 7)
```

Here is the corresponding output:

```
2025-04-27
2025-04-28
2025-04-29
2025-04-30
2025-05-01
2025-05-02
2025-05-03
2025-05-04
2025-05-05
2025-05-06
2025-05-07
```

Hint: Python datetime documentation is available here:
https://docs.python.org/3/library/datetime.html

***** The following exercise is optional, no need to submit a solution *****

9. Virtual chess board

Having acquired all the knowledge so far, it is important to put your skills to test. Let's build a virtual chessboard (command-line) for two human players.

a. A chessboard is a 8 * 8 matrix. It has a notation from A-H and 1-8. Here (1-8) are rows and (A-H) are columns. We always refer the position of a chess piece by its chess notation. For example, "E4" means 4th row, 5th column of the 8 * 8 matrix (see fig.1).

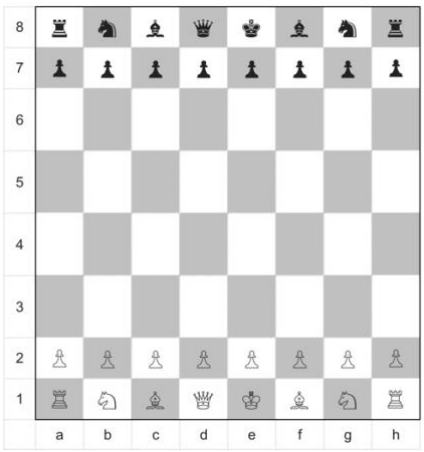

Figure 1: chess board

b. Your application should use exceptions to detect invalid moves. A chess move is valid, if the piece follows the rules (such as Knight moves in L shape) the target position of the move is not of the same colour and the piece does not end up outside of the board.

   Simple version of the chess rules is good enough for this exercise. For a list of all valid moves in chess, please refer to: https://en.wikipedia.org/wiki/Rules_of_chess

c. ***Hint***: You can display the Chess pieces either
   - Using letters: For example: black knight can be represented with symbol BK
   - Or even better, by using the unicode chess symbols. Here is a code snippet that displays all chess pieces in python:

```
pieces = '  '.join(chr(9812 + x) for x in range(12))
print(pieces)
```

   Running the above code will produce the following:

♔ ♕ ♖ ♗ ♘ ♙ ♚ ♛ ♜ ♝ ♞ ♟

   Note: Don't worry about colouring the squares. They can all be white/no-colour.

d. Your application should provide a '-help' option to display a brief textual guide on how to use your chess game.

e. You can write the code using object-oriented programming (classes/objects) but it's not mandatory

f. Try to play your game with a friend and make sure it behaves as expected. Code testing and validation is an important part of programming.

End.