REPUBLIC OF THE PHILIPPINES
POLYTECHNIC UNIVERSITY OF THE PHILIPPINES
STA. MESA, MANILA

COLLEGE OF ENGINEERING

ELECTRONICS ENGINEERING DEPARTMENT

CMPE 20022:

COMPUTER PROGRAMMING

INSTRUCTIONAL MATERIAL

MODULE 4 – LAB ACTIVITY

Module 4

Laboratory Activity # 1

Objectives

Familiarize the student with:

- projecting and writing parameterized functions;
- utilizing the `return` statement;
- testing the functions.

Scenario

Your task is to write and test a function which takes one argument (a year) and returns `True` if the year is a *leap year*, or `False` otherwise.

The seed of the function is already sown in the skeleton code in the editor.

Note: we've also prepared a short testing code, which you can use to test your function.

The code uses two lists - one with the test data, and the other containing the expected results. The code will tell you if any of your results are invalid.

```
1  def isYearLeap(year):
2  #
3  # put your code here
4  #
5
6  testData = [1900, 2000, 2016, 1987]
7  testResults = [False, True, True, False]
8  for i in range(len(testData)):
9      yr = testData[i]
10     print(yr,"->",end="")
11     result = isYearLeap(yr)
12     if result == testResults[i]:
13         print("OK")
14     else:
15         print("Failed")
```

Laboratory Activity # 2

Objectives

Familiarize the student with:

- projecting and writing parameterized functions;
- utilizing the `return` statement;
- utilizing the student's own functions.

Scenario

Your task is to write and test a function which takes two arguments (a year and a month) and returns the number of days for the given month/year pair (while only February is sensitive to the `year` value, your function should be universal).

The initial part of the function is ready. Now, convince the function to return `None` if its arguments don't make sense.

Of course, you can (and should) use the previously written and tested function (LAB 1). It may be very helpful. We encourage you to use a list filled with the months' lengths. You can create it inside the function - this trick will significantly shorten the code.

We've prepared a testing code. Expand it to include more test cases.

```
def isYearLeap(year):
#
# your code from LAB # 1
#

def daysInMonth(year, month):
#
# put your new code here
#

testYears = [1900, 2000, 2016, 1987]
testMonths = [2, 2, 1, 11]
testResults = [28, 29, 31, 30]
for i in range(len(testYears)):
    yr = testYears[i]
    mo = testMonths[i]
    print(yr, mo, "->", end="")
    result = daysInMonth(yr, mo)
    if result == testResults[i]:
        print("OK")
    else:
        print("Failed")
```

Laboratory Activity # 3

Objectives

Familiarize the student with:

- projecting and writing parameterized functions;
- utilizing the `return` statement;
- building a set of utility functions;
- utilizing the student's own functions.

Scenario

Your task is to write and test a function which takes three arguments (a year, a month, and a day of the month) and returns the corresponding day of the year, or returns `None` if any of the arguments is invalid.

Use the previously written and tested functions. Add some test cases to the code. This test is only a beginning.

```
1  def isYearLeap(year):
2  #
3  # your code from LAB 1
4  #
5
6  def daysInMonth(year, month):
7  #
8  # your code from LAB 2
9  #
10
11 def dayOfYear(year, month, day):
12 #
13 # put your new code here
14 #
15
16 print(dayOfYear(2000, 12, 31))
```

Laboratory Activity # 4

Objectives

- familiarizing the student with classic notions and algorithms;
- improving the student's skills in defining and using functions.

Scenario

*A natural number is **prime** if it is greater than 1 and has no divisors other than 1 and itself.*

Complicated? Not at all. For example, 8 isn't a prime number, as you can divide it by 2 and 4 (we can't use divisors equal to 1 and 8, as the definition prohibits this).

On the other hand, 7 is a prime number, as we can't find any legal divisors for it.

Your task is to write a function checking whether a number is prime or not.

The function:

- is called `isPrime`;
- takes one argument (the value to check)
- returns `True` if the argument is a prime number, and `False` otherwise.

Hint: try to divide the argument by all subsequent values (starting from 2) and check the remainder - if it's zero, your number cannot be a prime; think carefully about when you should stop the process.

If you need to know the square root of any value, you can utilize the `**` operator. Remember: the square root of `x` is the same as $x^{0.5}$

Complete the code in the editor.

```
def isPrime(num):
#
# put your code here
#

for i in range(1, 20):
    if isPrime(i + 1):
        print(i + 1, end=" ")
print()
```

Run your code and check whether your output is the same as ours.

Test Data

Expected output:

```
2 3 5 7 11 13 17 19
```

Laboratory Activity # 5

Objectives

- improving the student's skills in defining, using and testing functions.

Scenario

A car's fuel consumption may be expressed in many different ways. For example, in Europe, it is shown as the amount of fuel consumed per 100 kilometers.

In the USA, it is shown as the number of miles traveled by a car using one gallon of fuel.

Your task is to write a pair of functions converting l/100km into mpg, and vice versa.

The functions:

- are named `l100kmtompg` and `mpgtol100km` respectively;
- take one argument (the value corresponding to their names)

Complete the code in the editor.

```
def l100kmtompg(liters):
#
# put your code here
#

def mpgtol100km(miles):
#
# put your code here
#

print(l100kmtompg(3.9))
print(l100kmtompg(7.5))
print(l100kmtompg(10.))
print(mpgtol100km(60.3))
print(mpgtol100km(31.4))
print(mpgtol100km(23.5))
```

Run your code and check whether your output is the same as ours.

Here is some information to help you:

- 1 American mile = 1609.344 metres;
- 1 American gallon = 3.785411784 litres.

Test Data

Expected output:

```
60.31143162393162
31.36194444444444
23.52145833333333
3.9007393587617467
7.490910297239916
10.009131205673757
```

PROJECT

Objectives

- perfecting the student's skills in using Python for solving complex problems,
- integration of programming techniques in one program consisting of many various parts.

Scenario

Your task is to write **a simple program which pretends to play *tic-tac-toe* with the user**. To make it all easier for you, we've decided to simplify the game. Here are our assumptions:

- the computer (i.e., your program) should play the game using `'X'`s;
- the user (e.g., you) should play the game using `'O'`s;
- the first move belongs to the computer - it always puts its first `'X'` in the middle of the board;
- all the squares are numbered row by row starting with `1` (see the example session below for reference)
- the user inputs their move by entering the number of the square they choose - the number must be valid, i.e., it must be an integer, it must be greater than `0` and less than `10`, and it cannot point to a field which is already occupied;
- the program checks if the game is over - there are four possible verdicts: the game should continue, or the game ends with a tie, your win, or the computer's win;
- the computer responds with its move and the check is repeated;
- don't implement any form of artificial intelligence - a random field choice made by the computer is good enough for the game.

```
def DisplayBoard(board):
#
# the function accepts one parameter containing the board's current status
# and prints it out to the console
#

def EnterMove(board):
#
# the function accepts the board current status, asks the user about their move,
# checks the input and updates the board according to the user's decision
#

def MakeListOfFreeFields(board):
#
# the function browses the board and builds a list of all the free squares;
# the list consists of tuples, while each tuple is a pair of row and column numbers
#

def VictoryFor(board, sign):
#
# the function analyzes the board status in order to check if
# the player using 'O's or 'X's has won the game
#

def DrawMove(board):
#
# the function draws the computer's move and updates the board
```

The example session with the program may look as follows:

```
+-------+-------+-------+
|       |       |       |
|   1   |   2   |   3   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   4   |   X   |   6   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   7   |   8   |   9   |
|       |       |       |
+-------+-------+-------+
Enter your move: 1
+-------+-------+-------+
|       |       |       |
|   O   |   2   |   3   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   4   |   X   |   6   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   7   |   8   |   9   |
|       |       |       |
+-------+-------+-------+
+-------+-------+-------+
|       |       |       |
|   O   |   X   |   3   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   4   |   X   |   6   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   7   |   8   |   9   |
|       |       |       |
+-------+-------+-------+
Enter your move: 8
+-------+-------+-------+
|       |       |       |
|   O   |   X   |   3   |
|       |       |       |
+-------+-------+-------+
```

```
|       |       |       |
|   4   |   X   |   6   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   7   |   O   |   9   |
|       |       |       |
+-------+-------+-------+
+-------+-------+-------+
|       |       |       |
|   O   |   X   |   3   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   4   |   X   |   X   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   7   |   O   |   9   |
|       |       |       |
+-------+-------+-------+
Enter your move: 4
+-------+-------+-------+
|       |       |       |
|   O   |   X   |   3   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   O   |   X   |   X   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   7   |   O   |   9   |
|       |       |       |
+-------+-------+-------+
+-------+-------+-------+
|       |       |       |
|   O   |   X   |   X   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   O   |   X   |   X   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   7   |   O   |   9   |
|       |       |       |
+-------+-------+-------+
```

```
Enter your move: 7
+-------+-------+-------+
|       |       |       |
|   O   |   X   |   X   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   O   |   X   |   X   |
|       |       |       |
+-------+-------+-------+
|       |       |       |
|   O   |   O   |   9   |
|       |       |       |
+-------+-------+-------+
You won!
```

Requirements

Implement the following features:

- the board should be stored as a three-element list, while each element is another three-element list (the inner lists represent rows) so that all of the squares may be accessed using the following syntax:

  `board[row][column]`
- each of the inner list's elements can contain `'O'`, `'X'`, or a digit representing the square's number (such a square is considered free)
- the board's appearance should be exactly the same as the one presented in the example.
- implement the functions defined for you in the editor.

Drawing a random integer number can be done by utilizing a Python function called `randrange()`. The example program below shows how to use it (the program prints ten random numbers from 0 to 8).

Note: the `from-import` instruction provides an access to the `randrange` function defined within an external Python module callled `random`.

```
from random import randrange

for i in range(10):
    print(randrange(8))
```