CPS 2231 2025 Spring
Lab1: Review of CPS 1231

Instructor: Dr. Y. Tiffany Tang[1] and Dr. Pinata Winoto[2]
Songlin Shang[3], Cong Ye[4] and Zike Deng[5]

Instruction:

Self-check: In all labs in the future, you can download `check.py` in canvas, which will help you check the output format in labs, to make sure you don't have compile error or logic error before you submit your file. If you didn't finish installing the python environment/JDK installation, you can download the file from canvas named as tutorial.pdf. Please read and follow the tutorial, which teaches you how to install a python environment, for checking your code result. Also, the instructions of all questions also teach you how to use `check.py` to check your code.

Plagiarism: We encourage discussing all labs with your classmate, but **NEVER** copy other's code directly. We will use GradeScope[@] platform, to check the similarity probability, 90% similarity reported by GradeScope will be reviewed by TAs, and will report to your instructor and receive some penalty based on the event.

AI-Generate code issue: We encourage you to use AI tools, to help you compile all labs, **BUT NEVER** directly **copy and paste**. To detect that, we will use different tool to detect AI possibility include:

ChatGPT Code Detector: https://chatgpt.com/c/67bdddec-c728-8010-b1b6-abf3e56b3136

GPTSniffer: https://github.com/MDEGroup/GPTSniffer

Penalty will be given for high AI probability.

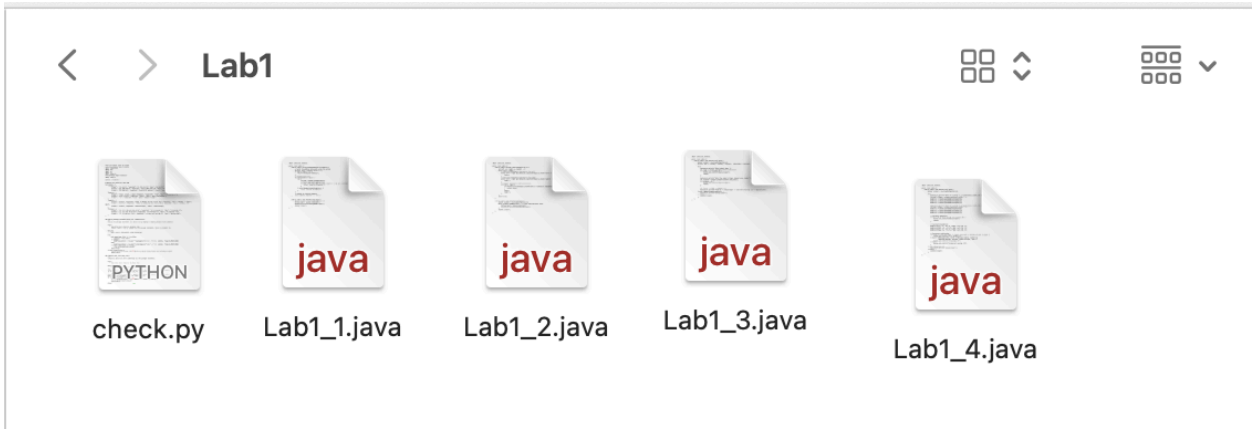
Grading: Each lab has the same points, and each test case has the same points. In check.py, you might check your code with given test cases in labs to avoid spelling errors and compile errors. We still have invisible test cases, which test for grading. This might test you to consider all possible positions. All test cases have the same points.

[1] Tiffany Ya Tang, Associate Professor, Department of Computer Science, Wenzhou-Kean University; Email: yatang@kean.edu
[2] Pinata Winoto, Assistant Professor, Department of Computer Science, Wenzhou-Kean University; Email: pwinoto@kean.edu
[3] Songlin Shang, Department of Computer Science, Wenzhou-Kean University; Email: shangs@kean.edu
[4] Cong Ye, Department of Computer Science, Wenzhou-Kean University; Email: yecon@kean.edu
[5] Zike Deng, Department of Computer Science, School of Computing and Data Science, University of Hong Kong; Email: baylordeng@connect.hku.hk

Get Started

Please download Lab1_1.java, Lab1_2.java, Lab1_3.java, Lab1_4.java on canvas.

Before you start the lab, you can create a project on eclipse called Lab1. Also, we suggest you download the check.py in canvas and move it under the same folder with all labs. This would be like:



Before you start coding, please direct you terminal to same folder using following code:

`cd path  (path is the way to the folder of check.py)`

here is an example:

```
(base) baylordeng@Mac ~ % cd /Users/baylordeng/Desktop/auto/Lab1/Lab1
(base) baylordeng@Mac Lab1 %
```

Now you can start coding after all these preparations.

Lab1_1

Write a method, which needs to handle a valid (IPv4) IP address as a String, print out a defanged version of that IP address. A defanged IP address replaces every period "." with "[.]".
Any Invalid value will get a response as "Invalid IP address."

Example 1:

Input: address = "1.1.1.1"
Output: "1[.]1[.]1[.]1"

Example 2:

Input: address = "255.100.50.0"
Output: "255[.]100[.]50[.]0"

Example 3:

Input: address = "255.100.0"
Output: "Invalid IP address."

Constraints:

The given address is a valid IPv4 address.

Self-Check:

When you finish coding, you can run following code:

```
python  check.py --lab Lab1_1
```

If everything good, you might see following content:

```
(base) baylordeng@Mac Lab1 % python check.py --lab Lab1_1
********* Compiling Java file: Lab1_1.java *********
✅ Compilation succeeded!
Running test case: Valid IP 1
********* Running Java program: Lab1_1 *********
✅ Execution succeeded!
✅ Output is correct: 1[.]1[.]1[.]1
Running test case: Valid IP 2
********* Running Java program: Lab1_1 *********
✅ Execution succeeded!
✅ Output is correct: 255[.]100[.]50[.]0
Running test case: Invalid IP
********* Running Java program: Lab1_1 *********
✅ Execution succeeded!
✅ Output is correct: Invalid IP address.
✅ All checks passed for the specified lab!
```

Lab 1_2

Using java, write a method, which can detect a phrase is a **palindrome** if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string s, return true *if it is a palindrome*, or false *otherwise*.

Example 1:
Input: s = "A man, a plan, a canal: Panama"
Output: true
Explanation: "amanaplanacanalpanama" is a palindrome.

Example 2:
Input: s = "race a car"
Output: false
Explanation: "raceacar" is not a palindrome.

Example 3:
Input: s = " "
Output: true
Explanation: s is an empty string "" after removing non-alphanumeric characters. Since an empty string reads the same forward and backward, it is a palindrome.

Constraints:

- 1 <= s.length <= 2 * 10^5
- s consists only of printable ASCII characters.

Self-Check:

When you finish coding, you can run following code:
`python  check.py --lab Lab1_2`
If everything good, you might see following content:

```
(base) baylordeng@Mac Lab1 % python check.py --lab Lab1_2
********* Compiling Java file: Lab1_2.java *********
✅ Compilation succeeded!
Running test case: Palindrome 1
********* Running Java program: Lab1_2 *********
✅ Execution succeeded!
✅ Output is correct: true
```

```
Running test case: Not a Palindrome
********* Running Java program: Lab1_2 *********
✅ Execution succeeded!
✅ Output is correct: false
Running test case: Empty String
********* Running Java program: Lab1_2 *********
✅ Execution succeeded!
✅ Output is correct: true
✅ All checks passed for the specified lab!
```

Lab 1_3

Write a program that prompts the user to enter an integer for today's day of the week (Sunday is 0, Monday is 1, ..., and Saturday is 6). Also, prompt the user to enter the number of days after today for a future day and display the future day of the week.

Example 1

Enter today's day: 1

Enter the number of days elapsed since today: 3

Today is Monday and the future day is Thursday

Example 2

Enter today's day: 0

Enter the number of days elapsed since today: 31

Today is Sunday and the future day is Wednesday

Example 3

Any Invalid input will lead the program stop and output "Invalid Input".

Self-Check:

When you finish coding, you can run following code:
`python  check.py --lab Lab1_3`
If everything good, you might see following content:

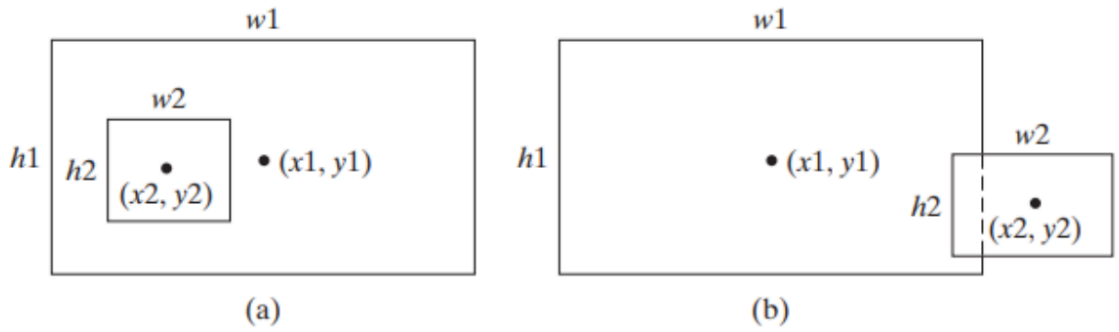
```
(base) baylordeng@Mac Lab1 % python check.py --lab Lab1_3

********* Compiling Java file: Lab1_3.java *********
✅ Compilation succeeded!
Running test case: Monday + 3 days
********* Running Java program: Lab1_3 *********
✅ Execution succeeded!
✅ Output is correct: Today is Monday and the future day is Thursday
Running test case: Sunday + 31 days
********* Running Java program: Lab1_3 *********
✅ Execution succeeded!
✅ Output is correct: Today is Sunday and the future day is Wednesday
Running test case: Invalid day
```

```
********* Running Java program: Lab1_3 *********
✅ Execution succeeded!
✅ Output is correct: Invalid Input
✅ All checks passed for the specified lab!
```

Lab1_4

Write a program that prompts the user to enter the center x-, y-coordinates, width, and height of two rectangles and determines whether the second rectangle is inside the first or overlaps with the first, as shown in Figure 3.9. Test your program to cover all cases.



Examples 1

Enter r1's center x-, y-coordinates, width, and height: 2.5 4 2.5 43

Enter r2's center x-, y-coordinates, width, and height: 1.5 5 0.5 3

Output: r2 is inside r1

Examples 2

Enter r1's center x-, y-coordinates, width, and height: 1 2 3 5.5

Enter r2's center x-, y-coordinates, width, and height: 3 4 4.5 5

Output: r2 overlaps r1

Examples 3

Enter r1's center x-, y-coordinates, width, and height: 1 2 3 3

Enter r2's center x-, y-coordinates, width, and height: 40 45 3 2

Output: r2 does not overlap r1

Self-Check:

When you finish coding, you can run following code:

```
python  check.py --lab Lab1_4
```

If everything good, you might see following content:

```
(base) baylordeng@Mac Lab1 % python check.py --lab Lab1_4
********* Compiling Java file: Lab1_4.java *********
✅ Compilation succeeded!
Running test case: r2 inside r1
********* Running Java program: Lab1_4 *********
✅ Execution succeeded!
✅ Output is correct: r2 is inside r1
Running test case: r2 overlaps r1
********* Running Java program: Lab1_4 *********
✅ Execution succeeded!
✅ Output is correct: r2 overlaps r1
Running test case: No overlap
********* Running Java program: Lab1_4 *********
✅ Execution succeeded!
✅ Output is correct: r2 does not overlap r1
✅ All checks passed for the specified lab!
```

Last Step

Congratulations! You have finished all Labs in Lab1.

Now please run the following code in the end:

```
python check.py --uid 123456
```

(replace 123456 as you student number)

```
(base) baylordeng@Mac Lab1 % python check.py --uid 123456
```

It will check all labs again and zip a file. If everythings good, you might see following information:

```
Your Student ID is 123456. Is this correct? (y/n):
```

Input "`y`", then press "`enter`", you will see following information:

```
********* Zipping Files *********
Zipping Lab1_1.java... Done.
Zipping Lab1_2.java... Done.
Zipping Lab1_3.java... Done.
Zipping Lab1_4.java... Done.

📦 Successfully created submission package: Lab1_123456_0226_1507.zip
```

You will see there is a new file under the folder like:

Submit this .zip file to canvas!