CMPT 120

Lecture 36 – Practice Exam 10 - **SOLUTION**

In-Class Activity

Course grading scheme on our course website: ***Best 7 in-class exercises out of 10: 1% each, for a total of 7%***

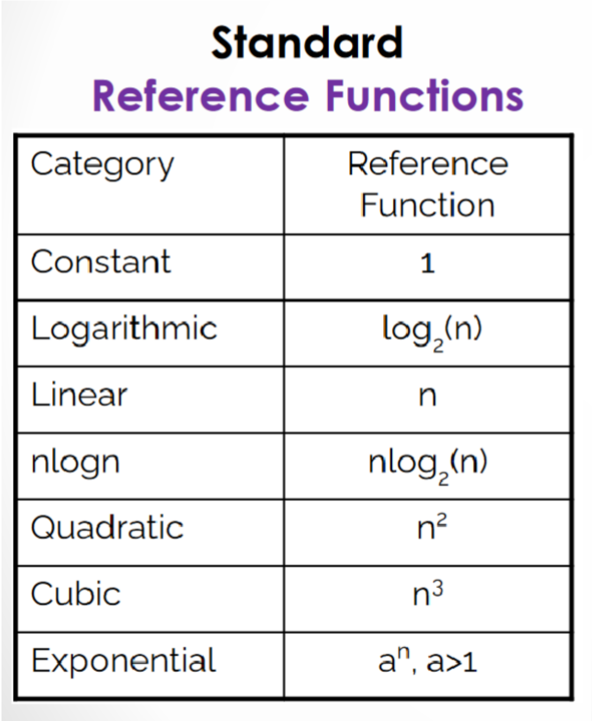
- Our in-class activity #10 -> 1%
  - Write your answer to questions 1, 2 and 7 on the provided sheet of paper
  - Write your **lastname**, **firstname** and **student number**
  - At the end of today's class, hand in your sheet of paper in the appropriate pile:
    - **Pile 1** -> if your lastname start with a letter that is between '**A**' and '**L**'
      - **Pile 1** is on your **left-hand side** of the classroom
    - **Pile 2** -> if your lastname start with a letter that is between '**M**' to letter '**Z**'
      - **Pile 2** is on your **right-hand side** of the classroom

Theory and Understanding

Try to answer the questions 1st **without using your computer**, then confirm your answer using IDLE!

How to analyze complexity

1. Count the number of times a **critical operation** is executed
   - Usually seen in a loop
   - Express this number as a function of **n** (number of elements) 0-> use Standard Refence functions:
2. Disregard **constants**
3. Disregard **lower exponent terms** (e.g., **n** when both **n**^{2} and **n** are present).

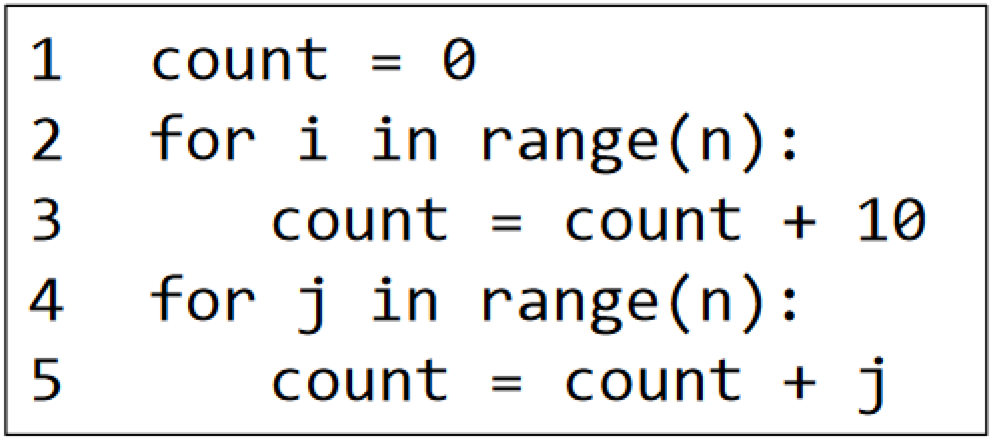


Standard
Reference Functions

Category	Reference Function
Constant	1
Logarithmic	$\log_2(n)$
Linear	n
nlogn	$n\log_2(n)$
Quadratic	n^2
Cubic	n^3
Exponential	a^n, a>1

Calculating Time Complexity – Example 1

- Critical operations depending on **n**?

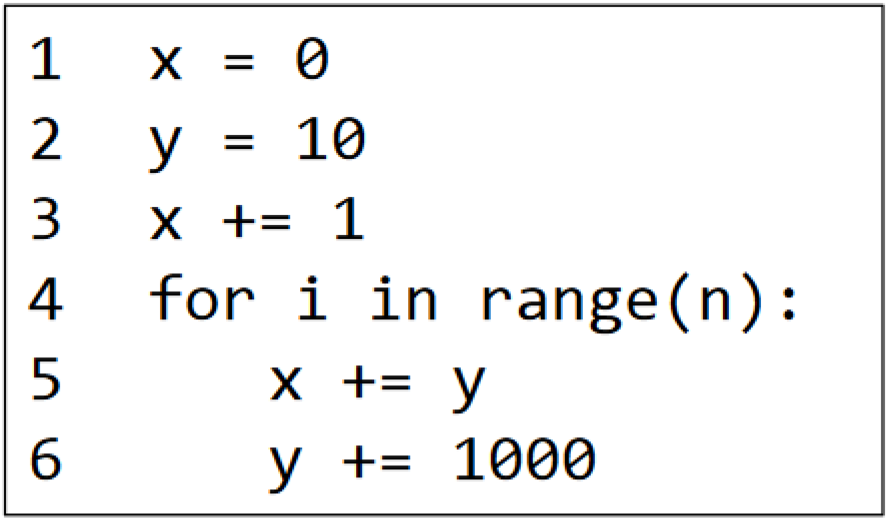


```
1   count = 0
2   for i in range(n):
3       count = count + 10
4   for j in range(n):
5       count = count + j
```

- How many additions are executed?
  - **n** + **n** = 2**n**

  (for loop at lines 2 and 3 repeated **n** times THEN for loop at lines 4 and 5 repeated **n** times)
- So, what is the order of this code fragment (its time complexity/efficiency)?
  - O(2**n**) -> 2 * O(**n**) -> O(**n**)
  - Can discard the factor “2”

Calculating Time Complexity – Question 1



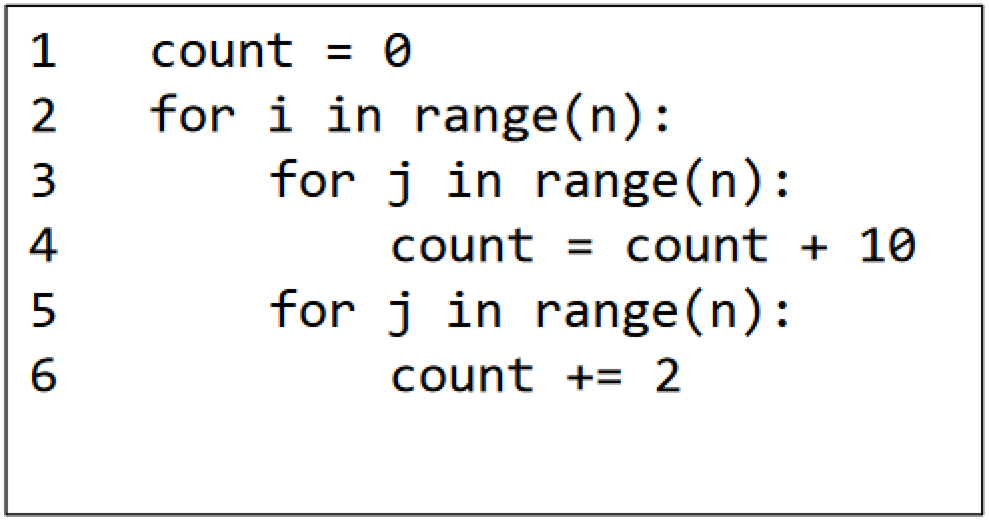
```
1   x = 0
2   y = 10
3   x += 1
4   for i in range(n):
5       x += y
6       y += 1000
```

- Critical operations depending on **n**?
  - Answer: Additions as well as assignments @ lines 5 and 6
- How many addition/assignments are executed?
  - Answer: 1 addition and 1 assignment executed at each iteration of the loop
  - There are n iterations of the loop, i.e., lines 5 and 6 repeated n times
  - Answer: 2 * n

- So, what is the order of this code fragment (its time complexity/efficiency)?
  - Answer: 2 * n = 2n and since 2 is a constant (also a factor), we discard it and get n which matches reference function n ☺. Then, we express this function using the big O notation: O(n)

Calculating Time Complexity – Example 2

- Critical operations depending on **n**?



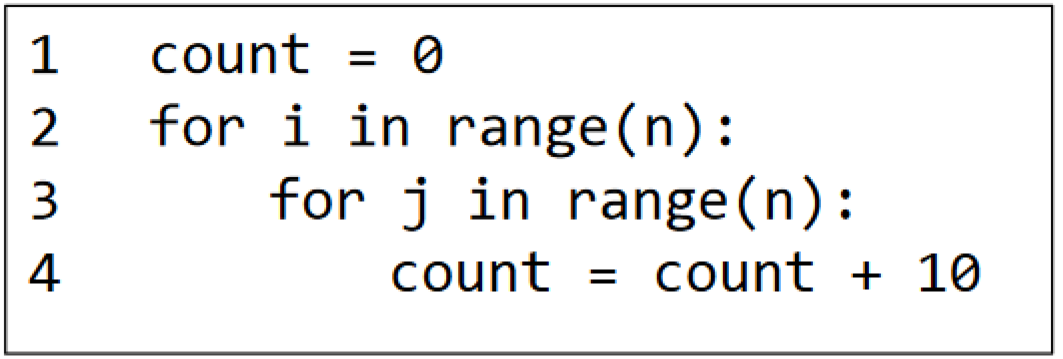
```
1    count = 0
2    for i in range(n):
3        for j in range(n):
4            count = count + 10
5        for j in range(n):
6            count += 2
```

- How many addition/assignments are executed?
  - (**n** + **n**) * **n** = 2**n** * **n** = 2$\mathbf{n}^2$

  (for loop at lines 3 and 4 repeated **n** times in 1st j loop THEN for loop at lines 5 and 6 repeated **n** times in 2nd j loop, both loops repeated **n** times in outer for loop)

- So, what is the order of this code fragment (its time complexity/efficiency)?
  - O(2$\mathbf{n}^2$) -> 2 * O($\mathbf{n}^2$) -> O($\mathbf{n}^2$)
  - Can discard the factor "2"

Calculating Time Complexity – Question 2

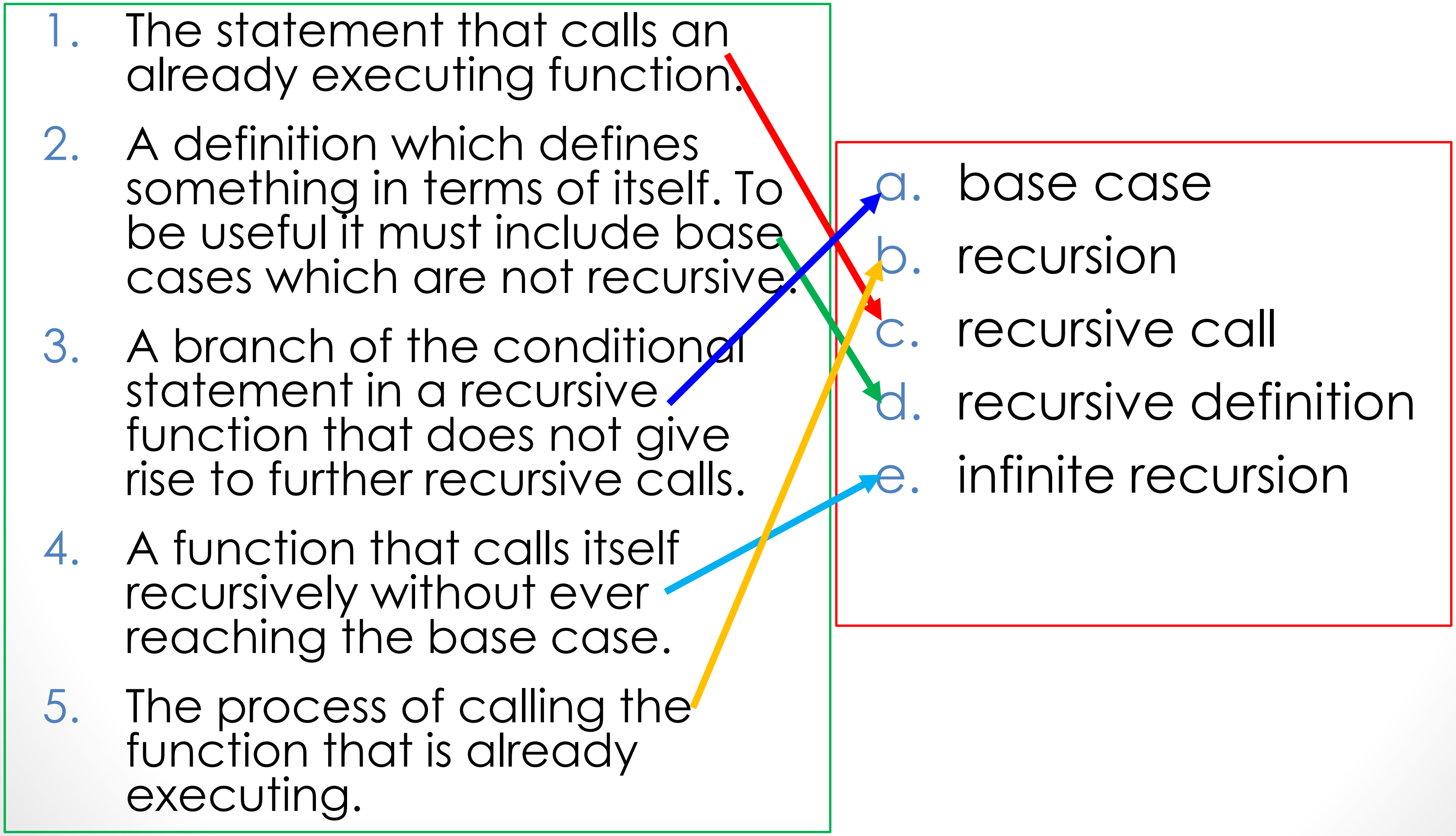


```
1    count = 0
2    for i in range(n):
3        for j in range(n):
4            count = count + 10
```

- Critical operations depending on **n**?
  - Answer: Additions as well as assignments @ line 4
- How many addition/assignments are executed?
  - Answer: 1 addition and 1 assignment executed at each iteration of the inner loop
  - There are n iterations of the inner loop
  - Then these n iterations are done at every iteration of the outer loop and there are n iterations of the outer loop
  - Answer: $(2 * n)n = 2n^2$
- So, what is the order of this code fragment (its time complexity/efficiency)?
  - Answer: Discarding 2, we get n^2 which matches reference function n^2 ☺. Lastly, we express this function using the big O notation: $O(n^2)$

Question 3 - Matching

Match each statement on the left with the most appropriate word(s) on the right.



1. The statement that calls an already executing function.
2. A definition which defines something in terms of itself. To be useful it must include base cases which are not recursive.
3. A branch of the conditional statement in a recursive function that does not give rise to further recursive calls.
4. A function that calls itself recursively without ever reaching the base case.
5. The process of calling the function that is already executing.

a. base case
b. recursion
c. recursive call
d. recursive definition
e. infinite recursion

Question 4 - Binary

1. How many distinct numbers can I represent with …
   a) 1 bit? 2 What are these numbers? 0 and 1
   b) 4 bits? $2^4 = 16$ What are these numbers (in binary)?
   0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111
2. How many distinct numbers can I represent with 7 bits? $2^7 = 128$

   These numbers range from __0__ to __127__
3. How many distinct numbers can I represent with 1 byte? 1 byte = 8 bits so $2^8 = 256$

   These numbers range from __0__ to __255__
4. How many distinct numbers can I represent with 32 bits? $2^{32} = 4,294,967,296$

   These numbers range from __0__ to _4,294,967,295_

Question 5 - Conversion

1. Convert 10011011 into an integer (decimal number):

1 x 2^7 + 1 x 2^4 + 1 x 2^3 + 1 x 2^1 + 1 x 2^0 = 128+16+8+2+1 = 155

2. a) What is the binary equivalent of 57?

57 – 32 = 25 – 16 = 9 – 8 = 1 – 1 = 0 => 00111001

2^5 2^4 2^3 2^0

b) What is the binary equivalent of 157?

157 – 128 = 29 – 16 = 13 – 8 = 5 – 4 = 1 – 1 = 0 => 10011101

2^7 2^4 2^3 2^2 2^0

Question 6

- In class, we learned a **Selection sort** algorithm that swapped the smallest number in a list with the first element @ index 0. **Selection sort** can also be implemented by selecting the largest number in the list, and swapping it with the last element @ index "len(list)-1".
- Using this ***updated*** algorithm, suppose you have the following list of numbers to sort:

  [11, 7, 12, 14, 19, 1, 6, 18, 8, 20]

  Which list below represents the partially sorted list after **three** complete iterations of **Selection sort**?

a. [7, 11, 12, 1, 6, 14, 8, 18, 19, 20]
b. [7, 11, 12, 14, 19, 1, 6, 18, 8, 20]
c. [11, 7, 12, 14, 1, 6, 8, 18, 19, 20]
d. [11, 7, 12, 14, 8, 1, 6, 18, 19, 20]
e. None of the above

Question 7

- Assume that a problem can be solved with two different algorithms, Algorithm **A** and Algorithm **B**, and you need to decide which algorithm to implement based on their time complexity.
- Algorithm **A** has a time complexity of O(**n**).
- Algorithm **B** has a time complexity of O(**n** $\log_2$ **n**).

- Which algorithm (Algorithm **A** or Algorithm **B**) would you choose, if you had a very large dataset, i.e., if **n** was very large?

- Answer: I would choose Algorithm A with time complexity of O(n) as it is "faster" (i.e., more time efficient) than Algorithm B

Question 8 – Test Cases

If we wanted to completely test this Python code fragment, how many test cases would we need, i.e., how many different `width` values must we enter? Note that we are looking at the minimum number of test cases.

What do we mean by “completely”? We mean that each of our test cases will execute a section of the program, such that all of our test cases will execute all statements in our program at least once.

```
width = int(input("Please, enter a width: "))
if width > 0 :
  if width > 10:
    if width % 2 == 0 :
      # then do something with width
    else :
      print(f"width {width} is not even.")
  else:
    print(f"0 < width {width} <= 10.")
else:
  print("width <= 0.")
```

[✓ **A.** 4]
B. 2
C. 1
D. 5
E. There are no test cases that could completely test the Python code fragment above.

Question 8 – Test Cases

If we wanted to completely test this Python code fragment, how many test cases would we need, i.e., how many different `width` values must we enter? Note that we are looking at the minimum number of test cases.

What do we mean by “completely”? We mean that each of our test cases will execute a section of the program, such that all of our test cases will execute all statements in our program at least once.

```
width = int(input("Please, enter a width: "))Always executed!
if width > 0 :  True
  if width > 10: True
    if width % 2 == 0 :True
      # then do something with width
    else :
      print(f"width {width} is not even.")
  else:
    print(f"0 < width {width} <= 10.")
else:
  print("width <= 0.")
```

Test case 1: width = 20

[✓ **A.** 4]
B. 2
C. 1
D. 5
E. There are no test cases that could completely test the Python code fragment above.

Question 8 – Test Cases

If we wanted to completely test this Python code fragment, how many test cases would we need, i.e., how many different `width` values must we enter? Note that we are looking at the minimum number of test cases.

What do we mean by "completely"? We mean that each of our test cases will execute a section of the program, such that all of our test cases will execute all statements in our program at least once.

```
width = int(input("Please, enter a width: "))Always executed!
if width > 0 : True
  if width > 10: True
    if width % 2 == 0 : False
      # then do something with width
    else :
      print(f"width {width} is not even.")
  else:
    print(f"0 < width {width} <= 10.")
else:
  print("width <= 0.")
```

Test case 2: width = 21

[✓ **A.** 4]
B. 2
C. 1
D. 5
E. There are no test cases that could completely test the Python code fragment above.

Question 8 – Test Cases

If we wanted to completely test this Python code fragment, how many test cases would we need, i.e., how many different `width` values must we enter? Note that we are looking at the minimum number of test cases.

What do we mean by "completely"? We mean that each of our test cases will execute a section of the program, such that all of our test cases will execute all statements in our program at least once.

```
width = int(input("Please, enter a width: "))Always executed!
if width > 0 : True
  if width > 10: False
    if width % 2 == 0 :
      # then do something with width
    else :
      print(f"width {width} is not even.")
  else:
    print(f"0 < width {width} <= 10.")
else:
  print("width <= 0.")
```

Test case 3: width = 9

[✓ **A.** 4]
B. 2
C. 1
D. 5
E. There are no test cases that could completely test the Python code fragment above.

Question 8 – Test Cases

If we wanted to completely test this Python code fragment, how many test cases would we need, i.e., how many different `width` values must we enter? Note that we are looking at the minimum number of test cases.

What do we mean by “completely”? We mean that each of our test cases will execute a section of the program, such that all of our test cases will execute all statements in our program at least once.

```
width = int(input("Please, enter a width: "))Always executed!
if width > 0 : False
  if width > 10:
    if width % 2 == 0 :
      # then do something with width
    else :
      print(f"width {width} is not even.")
  else:
    print(f"0 < width {width} <= 10.")
else:
  print("width <= 0.")
```

Test case 4: width = -3

[✓ **A.** 4]
B. 2
C. 1
D. 5
E. There are no test cases that could completely test the Python code fragment above.

Question 8 – Test Cases

If we wanted to completely test this Python code fragment, how many test cases would we need, i.e., how many different `width` values must we enter? Note that we are looking at the minimum number of test cases.

What do we mean by “completely”? We mean that each of our test cases will execute a section of the program, such that all of our test cases will execute all statements in our program at least once.

```
width = int(input("Please, enter a width: "))Always executed!
if width > 0 : False
  if width > 10:                                Test case 5: width = 0
    if width % 2 == 0 :
      # then do something with width
    else :
      print(f"width {width} is not even.")
  else:
    print(f"0 < width {width} <= 10.")
else:
  print("width <= 0.")
```

[✓ **A.** 4]
B. 2
C. 1
D. 5
E. There are no test cases that could completely test the Python code fragment above.

A 5th test case could be **width = 0**, but this 5th test case would execute the same statements as the statements executed by **test case 4** as illustrated on the previous slide! Verify for yourself! And since we are looking for the minimum number of test cases, then 4 test cases is the answer!

Question 9

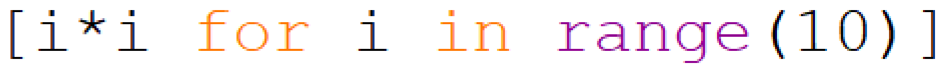
- Translate the following message:

```
01000111 01101111 01101111 01100100
00100000 01101100 01110101 01100011
01101011 00100000 01101001 01101110
00100000 01111001 01101111 01110101
01110010 00100000 01100110 01101001
01101110 01100001 01101100 00100000
01100101 01111000 01100001 01101101
01110011 00100001 0001010
```

- Answer: Good luck in your final exams!

Question 10 – List Comprehension

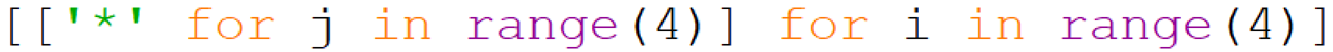
1. What would this code fragment produce?



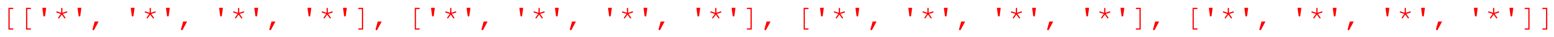
```
[i*i for i in range(10)]
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

2. What would this code fragment produce?



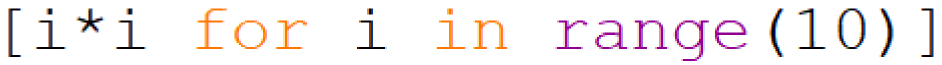
```
[['*' for j in range(4)] for i in range(4)]
```



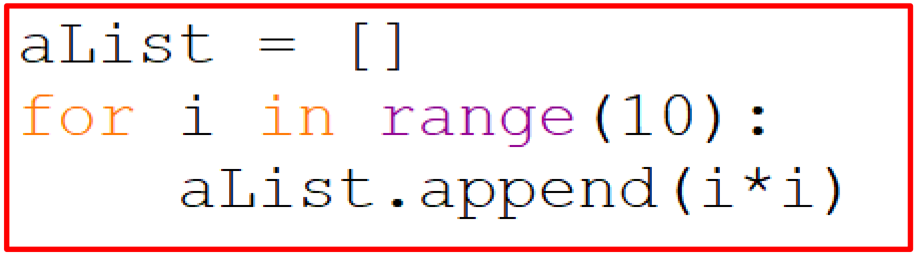
```
[['*', '*', '*', '*'], ['*', '*', '*', '*'], ['*', '*', '*', '*'], ['*', '*', '*', '*']]
```

Question 10 (cont'd)

3. Rewrite the code fragment below into a for loop such that both code fragments (the one below and your loop) produce the same result.



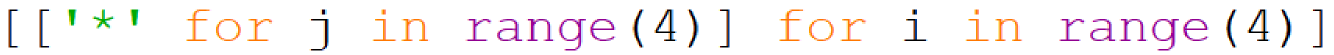
```
[i*i for i in range(10)]
```



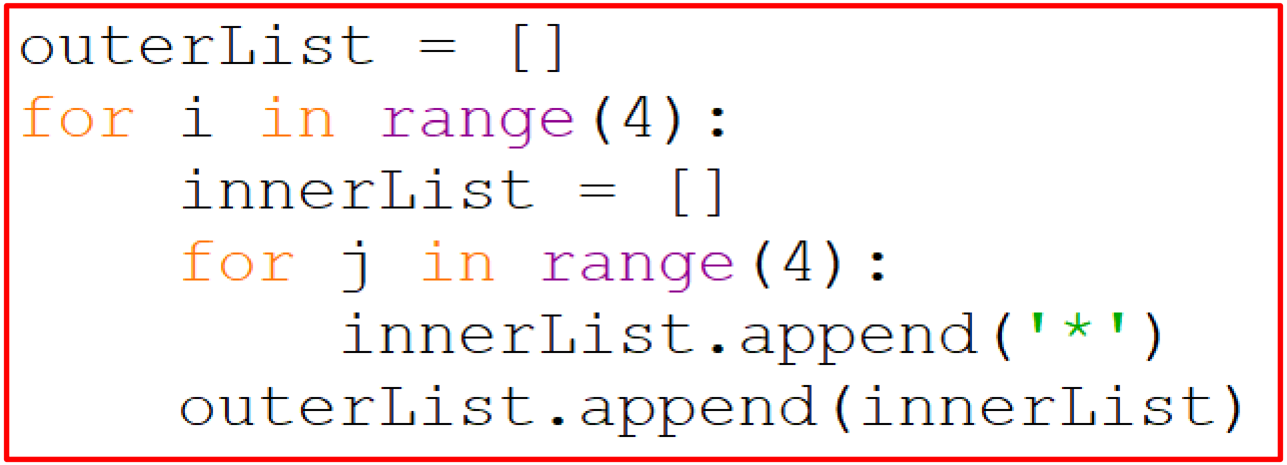
```
aList = []
for i in range(10):
    aList.append(i*i)
```

Question 10 (cont'd)

4. Rewrite the code fragment below into for loops such that both code fragments (the one below and your loops) produce the same result.



```
[['*' for j in range(4)] for i in range(4)]
```



```
outerList = []
for i in range(4):
    innerList = []
    for j in range(4):
        innerList.append('*')
    outerList.append(innerList)
```

Coding

Try to solve the problem (i.e., write your Python program) **1st on a piece of paper without using your computer**!

Question 1 - Searching

Step 1 - Problem Statement

- Given a list of integers and a target, write a search function that will return a list containing all the indices where the target can be found in the list. If it cannot be found, return an empty list.

Requirements

- Your solution must use the **`append`** function.

Step 4 – Testing

- You must write at least 3 test cases.

Question 2 - Vowel Counter

Step 1 - Problem Statement

- Write a function called **`countVowels(aString,vowelCount)`** that returns the number of vowels in the string **`aString`** using recursion.

Question 3 - Conversion

Step 1 - Problem Statement

Write a function that converts a given binary number (entered as a string) into a decimal number and returns it.

Question 4 - Conversion

Step 1 - Problem Statement

Write a function that converts a given decimal number (entered as an integer) into a binary number and returns it.