CSC 110 Midterm 2 Study Guide

Question 1

Loop table

Give the function below:

```
def double_nested(lists):
  for i in range(len(lists)):
    for j in range(len(lists[i])):
      lists[i][j] *= 2
  return lists
```

Complete the `loop table` below with the corresponding values of `i`, `j`, `len(lists[i])` and `lists` for the following function call:

```
double_nested([[0, 1, 3, 1], [2]])
```

RESPONSE FOR QUESTION 1:

i	j	len(lists[i])	lists
			[[_____, _____, _____, _____], [_____]]
			[[_____, _____, _____, _____], [_____]]
			[[_____, _____, _____, _____], [_____]]
			[[_____, _____, _____, _____], [_____]]
			[[_____, _____, _____, _____], [_____]]

Question 2

Loop table

Give the function below:

```
def reverse_strings_nested(strings):
  for i in range(len(strings)):
    for j in range(len(strings[i])):
      strings[i][j] = strings[i][j][::-1]
  return strings
```

Complete the `loop table` below with the corresponding values of `i`, `j`, `len(lists[i])` and `lists` for the following function call:

```
reverse_strings_nested([["war", "peek"], ["dog"], ["pets", "snug", "net"]])
```

RESPONSE FOR QUESTION 2:

i	j	len(strings[i])	strings[i][j]	strings[i][j][::-1]	strings
					[[_____, _____], [_____], [_____, _____, _____]]
					[[_____, _____], [_____], [_____, _____, _____]]
					[[_____, _____], [_____], [_____, _____, _____]]
					[[_____, _____], [_____], [_____, _____, _____]]
					[[_____, _____], [_____], [_____, _____, _____]]
					[[_____, _____], [_____], [_____, _____, _____]]

Question 3

Loop table

Give the function below:

```
def nested_min(lists):
  min = None
  for i in range(len(lists)):
    for j in range(len(lists[i])):
      if min == None or lists[i][j] < min:
        min = lists[i][j]
  return min
```

Complete the `loop table` below with the corresponding values of `i`, `j`, `len(lists[i])`, `lists[i][j]` and `min` for the following function call:

```
nested_min([[4, 3, 1, 0], [], [7, 2, 1]])
```

RESPONSE FOR QUESTION 3:

i	j	len(lists[i])	lists[i][j]	min

Lists

Evaluate the code

Evaluate the code below (when the code throws an error, the answer should be `ERROR`)

```
numbers = [1, 2]
numbers.append(5)
numbers.append(0)
numbers
```

```
numbers = [1, 2]
numbers.remove(1)
numbers
```

```
numbers = [1, 2]
numbers.pop(1)
numbers
```

```
numbers = [1, 2]
numbers[1]
numbers
```

```
numbers = [2, 3, 20, 1, 2, 40, 2]
numbers[7]
```

```
numbers = [2, 3, 20]
numbers[3] = 10
numbers
```

Dictionaries

Evaluate the code

Evaluate the code below (when the code throws an error, the answer should be `ERROR`)

```
counts = {"a": 1, "b": 2}
counts[1] = "c"
counts
```

```
counts = {"a": 1, "b": 2}
counts["a"] = "c"
counts
```

```
counts = {"a": 1}
counts.append("b": 2)
counts
```

Reading code

Shown below is a description of a function. After the description you will be shown three functions. You should select which of the three is the correct implementation. For the other two, identify and explain at least one issue with each one that would either cause an error, or cause the wrong answer to be produced.

The description of the function is as follows:

Given a list of years, return `True` if all years in the list are leap years.

Leap years are either (check **both** cases):

- divisible by 4 and not divisible by 100

OR

- divisible by 400

For example:

```
assert leap_year_check([2000, 2004, 2008, 2012, 2016, 2020, 2024]) == True
assert leap_year_check([2001, 2000, 2010, 2018]) == False
```

The three options are below:

A

```
1 def leap_year_check(years):
2    index = 0
3    while index < len(years):
4        if years[index] % 4 == 0 and years[index] % 100 != 0:
5            return True
6        elif years[index] % 400 == 0:
7            return True
8        index += 1
9    return False
```

B

```
1 def leap_year_check(years):
2    index = 0
3    while index < len(years):
4        if years[index] % 4 != 0 and years[index] % 400 != 0:
5            return False
6        index += 1
7    return True
```

C

```
1 def leap_year_check(years):
2    index = 0
3    while index < len(years):
4        if years[index] % 4 == 0 and years[index] % 100 != 0:
5            return False
6        elif years[index] % 400 == 0:
7            return False
8    index += 1
9    return True
```

KEY

Question 1

Loop table

Give the function below:

```
def double_nested(lists):
  for i in range(len(lists)):
    for j in range(len(lists[i])):
      lists[i][j] *= 2
  return lists
```

Complete the `loop table` below with the corresponding values of `i`, `j`, `len(lists[i])` and `lists` for the following function call:

```
double_nested([[0, 1, 3, 1], [2]])
```

RESPONSE FOR QUESTION 1:

i	j	len(lists[i])	lists
0	0	4	[[0, 1, 3, 1], [2]]
0	1	4	[[0, 2, 3, 1], [2]]
0	2	4	[[0, 2, 6, 1], [2]]
0	3	4	[[0, 2, 6, 2], [2]]
1	0	1	[[0, 2, 6, 2], [4]]

Question 2

Loop table

Give the function below:

```
def reverse_strings_nested(strings):
  for i in range(len(strings)):
    for j in range(len(strings[i])):
      strings[i][j] = strings[i][j][::-1]
  return strings
```

Complete the `loop table` below with the corresponding values of `i`, `j`, `len(lists[i])` and `lists` for the following function call:

```
reverse_strings_nested([["war", "peek"], ["dog"], ["pets", "snug", "net"]])
```

RESPONSE FOR QUESTION 2:

i	j	len(strings[i])	strings[i][j]	strings[i][j][::-1]	strings
0	0	2	war	raw	[["raw", "peek"], ["dog"], ["pets", "snug", "net"]]
0	1	2	peek	keep	[["raw", "keep"], ["dog"], ["pets", "snug", "net"]]
1	0	1	dog	god	[["raw", "keep"], ["god"], ["pets", "snug", "net"]]
2	0	3	pets	step	[["raw", "keep"], ["god"], ["step", "snug", "net"]]
2	1	3	snug	guns	[["raw", "keep"], ["god"], ["step", "guns", "net"]]
2	2	3	net	ten	[["raw", "keep"], ["god"], ["step", "guns", "ten"]]

Question 3

Loop table

Give the function below:

```
def nested_min(lists):
  min = None
  for i in range(len(lists)):
    for j in range(len(lists[i])):
      if min == None or lists[i][j] < min:
        min = lists[i][j]
  return min
```

Complete the `loop table` below with the corresponding values of `i`, `j`, `len(lists[i])`, `lists[i][j]` and `min` for the following function call:

```
nested_min([[4, 3, 1, 0], [], [7, 2, 1]])
```

RESPONSE FOR QUESTION 3:

i	j	len(lists[i])	lists[i][j]	min
0	0	4	4	4
0	1	4	3	3
0	2	4	1	1
0	3	4	0	0
1	-	0	-	0
2	0	3	7	0
2	1	3	2	0
2	2	3	1	0

Lists

Evaluate the code

Evaluate the code below (when the code throws an error, the answer should be `ERROR`)

```
numbers = [1, 2]
numbers.append(5)
numbers.append(0)
numbers
```

```
## [1, 2, 5, 0]
```

```
numbers = [1, 2]
numbers.remove(1)
numbers
```

```
## [2]
```

```
numbers = [1, 2]
numbers.pop(1)
```

```
## 2
```

```
numbers
```

```
## [1]
```

```
numbers = [1, 2]
numbers[1]
```

```
## 2
```

```
numbers
```

```
## [1, 2]
```

```
numbers = [2, 3, 20, 1, 2, 40, 2]
numbers[7]
```

ERROR

```
numbers = [2, 3, 20]
numbers[3] = 10
numbers
```

ERROR

Dictionaries

Evaluate the code

Evaluate the code below (when the code throws an error, the answer should be `ERROR`)

```
counts = {"a": 1, "b": 2}
counts[1] = "c"
counts
```

```
## {'a': 1, 'b': 2, 1: 'c'}
```

```
counts = {"a": 1, "b": 2}
counts["a"] = "c"
counts
```

```
## {'a': 'c', 'b': 2}
```

```
counts = {"a": 1}
counts.append("b": 2)
counts
```

ERROR

Reading code

Correct solution is solution **B**

Solution A will return True when it finds the first year that is leap.

Solution C has an infinite loop in line 8, and if that were fixed, it would return False when it finds the first non-leap year.