Polytechnic Tutoring Center

Exam 2 Review - CS 1113, Spring 2021

Disclaimer: This mock exam is only for practice. It was made by tutors in the Polytechnic Tutoring Center and is not representative of the actual exam given by the CS Department.

1. Determine the output of each of the following. If the code results in an error, write ERROR. If the output is None, write NONE.

(a)
```
my_str = ‘the quick brown fox’
print(my_str.split())
```

(b)
```
my_tup = (0, 10)
my_tup[1] = 20
print(my_tup[1])
```

(c)
```
lst = [2, 4, 6, 8, 10]
print(lst.reverse())
```

(d)
```
my_str = ‘school is cool’
print(my_str[8:1:-2])
```

(e)
```
a = ‘a’
b = ‘b’
print(3*a + 5*b)
```

(f)
```
lst = [1, 3, 5, 7, 9, 11]
lst.insert(5, 3)
print(lst)
```

(g)
```
lst = [1, 3, 5, 7, 9, 11]
print(lst.pop())
```

(h)
```
my_str = ‘RacECar’
print(my_str.upper())
```

2. Write a function called `row_count` that, when given an integer, n, will produce n rows of numbers, each counting either up to or down from n – i, where i, represents the row number starting from the 0th row. When i is even, count up to n. When i is odd, count down from n - i. An example of the desired output is provided below:

```
>>> row_count(5)

1 2 3 4 5
4 3 2 1
1 2 3
2 1
1
```

3. Write a function that identifies whether the input string is a palindrome. A palindrome is a word that reads the same backwards as when read normally. For example, ‘racecar’ is a palindrome. Note that the algorithm should not distinguish between upper and lower case letters. For the purposes of this exercise, we will consider them to be the same. That is ‘rAcEcar’ would still be a palindrome.

4. What is the output of the following code?

```
def func(symbol, n):
   for i in range(n):
         spaces = “ “*(n-i-1)
         fill = symbol*(2*i+1)
         print(spaces+fill)

func(‘*’, 5)
```

5. The exponential function e^x has an equivalent expansion as shown below:

$$e^x = 1 + x + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + \cdots = \sum_{k=0}^{\infty} a_k \ , \ \ where\ a_k = \frac{x^k}{k!}$$

Write a program that first takes in a number from the user, any number. Then, ask the user for more input, indicating whether to continue. If the user enters -1, the program should stop. Note that if the user enters -1 in the first iteration, the program should simply output 1. Now, for as long as the user does not enter -1, add the subsequent term for the expansion of e^x, to the result. When he enters -1, print out the resultant approximation of the function e^x, as well as a list of the approximations after each iteration. A few examples are provided below:

```
>> Give a number, x: 1
Stop? -1
Result is: 1
Sequence: [1]

>> Give a number, x: 1
Stop? No
Stop? -1
Result is: 2
Sequence is: [1, 2]

>> Give a number, x: 1
Stop? No
Stop? No
Stop? -1
Result is: 2.5
Sequence is: [1, 2, 2.5]
```