Facit och kommentarer till tenta 2024-03-07 i DA2004 och DA2005

Del A

1. A, B, C. After realizing that xor exists in the module ”operator”, I am also accepting E as correct answer, even though it is meant as a bitwise operator.

2. A

3. C

4. E

5. E

6. D

7. C

8. A

Del B: kodfrågor

9.
   - The return value is a list, not a string.
   - The i variable is global. Python will actually signal an error because a global variable is assigned without having a `global i` statement. But even if one got that right, the function will not function correctly in subsequent calls.

10.
```
def rev(string_to_reverse):
    '''
    Return the input string, but reversed.
    '''
    if s == '':
        raise ValueError('Empty string given to rev')
    reversed_s = ''
    for char in string_to_reverse:
        reversed_s = char + reversed_s
    return reversed_s
```

11. First suggestion, a compact function using list comprehension:

```
def monotonic(lst):
    lst1 = lst[:-1]
    lst2 = lst[1:]
    diffs = [ e1 - e2 for  (e1, e2) in zip(lst1, lst2)]
    return all(diff >= 0 for diff in diffs) or all(diff <= 0 for diff in diffs)
```

Second suggestion, using a function in the standard library:

```
def monotonic2(lst):
    if lst == sorted(lst):
        return True
    elif lst == sorted(lst, reverse=True):
        return True
    else:
        return False
```

And a third suggestion, simpler and perhaps more straightforward:

```
def increasing(lst):
    for i in range(len(lst)-1):
        if lst[i] > lst[i+1]:
            return False
    return True

def decreasing(lst):
    for i in range(len(lst)-1):
```

```
        if lst[i] < lst[i+1]:
            return False
    return True

def monotonic3(lst):
    if increasing(lst) or decreasing(lst):
        return True
    else:
        return False
```

To make this solution more ”self containing”, one can make the increasing and decreasing functions internal to monotonic3.

12.
```
def loan_analysis(initial_loan, monthly_payment, yearly_interest):
    loan = initial_loan
    month = 0
    while loan > monthly_payment:
        month += 1
        interest = round(loan * yearly_interest / 12, 2)
        loan = round(loan + interest - monthly_payment, 2)
        print(month, loan, interest)
```

13.
```
class Table:
    def __init__(self):
        self._ldict = {}
        self._rdict = {}

    def add(self, key, val):
        self._ldict[key] = val
        self._rdict[val] = key

    def items(self):
        return self._ldict.items()

    def get(self, key):
        if key in self._ldict:
            return self._ldict[key]
        elif key in self._rdict:
            return self._rdict[key]
        else:
            raise KeyError(f"'{key}' is not found in the table")

    def __repr__(self):
        return str(self)
    def __str__(self):
        return f'<Table {str([(a, b) for a, b in self._ldict.items()])}>'
```