Lecture 13: Summary

Morten Rieger Hannemose, Vedrana Andersen Dahl

Fall 2023

Important announcements

- If you have not received a mail from us, you are ok for the exam.
- DE test – midterm, but with updated instructions.
- Python Installation Support.
- Examenscafe, søndag 3/12 9-16.

Exam - sumup

- 6th of December, lokale på eksamensplan.dtu.dk
- 4 hours (extended time handled by AUS)
- No internet.
- No Large Language Models (ChatGPT etc.)
- Yes to website download, or download of any other text.
- Just like midterm.
- Hand in: Token file + python files
- Your grade based only on tests (provided and hidden)

Exam - grading

Remember (added to the first-page text)

If you believe there is a mistake or ambiguity in the text, you should use the most reasonable interpretation of the text to solve the task to the best of your ability. If we, after the exam, find inconsistencies in one or more tasks, this will be taken into account in the assessment.

But: we will not be fixing your code!

Exam - tips

Format

- 10 tasks
- All weeks
- Equal weight
- Unordered

Don't get stuck; move on

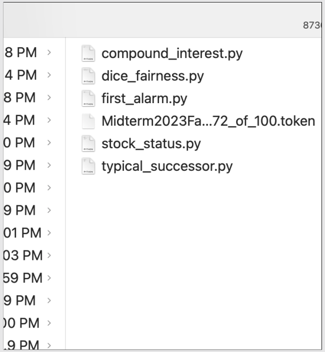
Difficulty

- Compound interest
- Stock Status
- First Alarm
- Typical Successor
- Dice Fairness

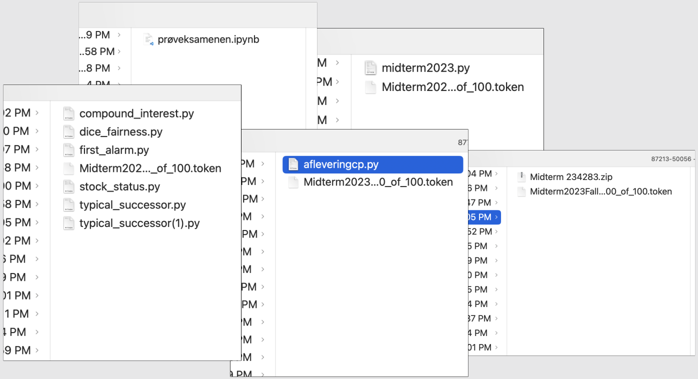
Identify easy tasks; skip difficult ones. When solving a task, read the entire text first.

Hand-in

Yes


compound_interest.py
dice_fairness.py
first_alarm.py
Midterm2023Fa...72_of_100.token
stock_status.py
typical_successor.py

No


prøveksamenen.ipynb

midterm2023.py
Midterm202...of_100.token

compound_interest.py
dice_fairness.py
first_alarm.py
Midterm202..._of_100.token
stock_status.py
typical_successor.py
typical_successor(1).py

afleveringcp.py
Midterm2023...0_of_100.token

Midterm 234283.zip
Midterm2023Fall...00_of_100.token

Hand in: token file and the python files with your solution, named as downloaded.

Screenshots we show

- Inspired by or copied from actual submissions.
- Shows *common* mistakes – you are not alone.

Inputs

You can assume that the input is as described. For example, dice fairness.

dice_fairness.py

You want to check if a set of dice throws is fair. Your task is to write a function that takes as input a list of numbers from 1 to 6 and returns:

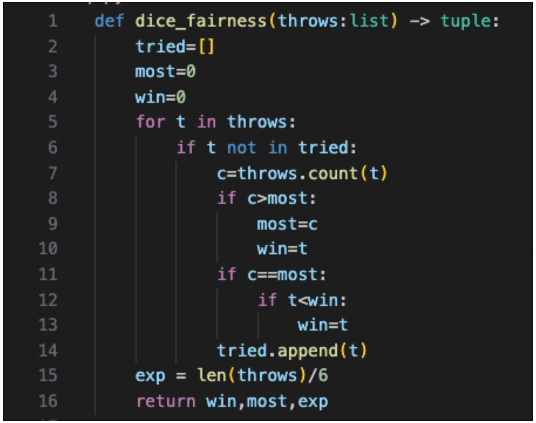
- The number that appears most frequently among the throws.
- The number of times this most frequent number appears.
- The expected number of times for a number to be thrown, calculated by dividing the number of throws by 6.

You can assume that the input list is not empty and contains only numbers from 1 to 6. If there is a tie for the most frequent number, the function should return the smaller number.

```
throws = [4, 2, 4, 4, 5, 6, 1, 2, 3, 4, 2, 3, 5, 5, 4, 4, 3, 2, 1, 4, 6]
```

Inputs

You can assume that the input is as described. For example, here is a very careful (correct) solution for dice fairness.



```python
def dice_fairness(throws:list) -> tuple:
    tried=[]
    most=0
    win=0
    for t in throws:
        if t not in tried:
            c=throws.count(t)
            if c>most:
                most=c
                win=t
            if c==most:
                if t<win:
                    win=t
            tried.append(t)
    exp = len(throws)/6
    return win,most,exp
```

It's ok to say:

```python
for i in range(1, 7):
    # count all occurrences of i
```

or

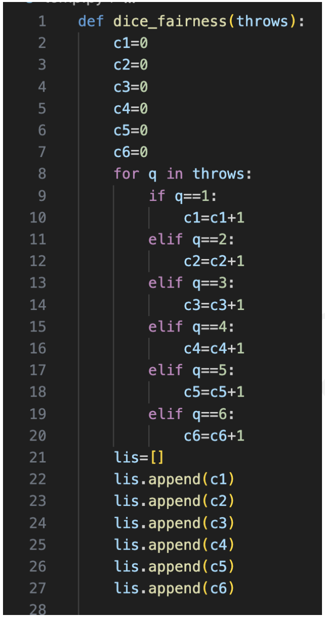
```python
counts = {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0}
```

or

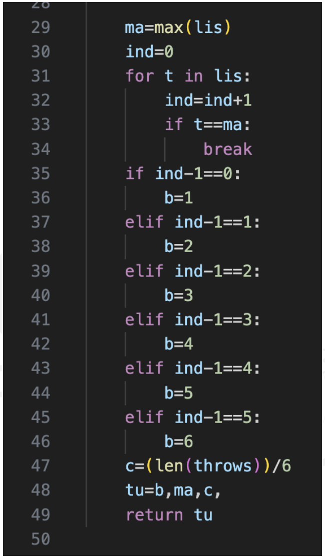
```python
counts = [0, 0, 0, 0, 0, 0]
for t in throws:
    counts[t-1] += 1
```

Inputs

This works correctly and follows a valid idea for a solution. However, with many lines of code, you might miss spotting an error.



```
def dice_fairness(throws):
    c1=0
    c2=0
    c3=0
    c4=0
    c5=0
    c6=0
    for q in throws:
        if q==1:
            c1=c1+1
        elif q==2:
            c2=c2+1
        elif q==3:
            c3=c3+1
        elif q==4:
            c4=c4+1
        elif q==5:
            c5=c5+1
        elif q==6:
            c6=c6+1
    lis=[]
    lis.append(c1)
    lis.append(c2)
    lis.append(c3)
    lis.append(c4)
    lis.append(c5)
    lis.append(c6)

```



```
    ma=max(lis)
    ind=0
    for t in lis:
        ind=ind+1
        if t==ma:
            break
    if ind-1==0:
        b=1
    elif ind-1==1:
        b=2
    elif ind-1==2:
        b=3
    elif ind-1==3:
        b=4
    elif ind-1==4:
        b=5
    elif ind-1==5:
        b=6
    c=(len(throws))/6
    tu=b,ma,c,
    return tu

```

Special conditions

Be careful about the first, last, or anyhow special.

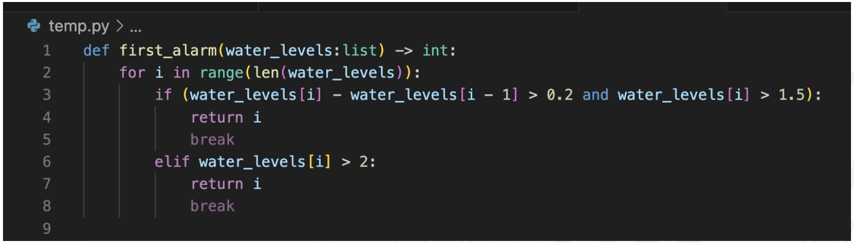
`first_alarm.py`

The water level of a river is measured (in meters) and recorded every hour. An alarm is triggered if any of the following two conditions are met:

1. The water level has risen by more than 0.2 meters during the last hour, and the resulting water level is higher than 1.5 meters.
2. The water level is above 2.0 meters.

Your task is to write a function that returns the index of the first alarm. If no alarm is triggered, the function should return -1. All inequalities are strict for example, a water level of exactly 2.0 meters is not enough to trigger an alarm. As an example, consider the input: `water_levels = [1.52, 1.29, 1.32, 1.18, 1.45, 1.63, 1.81, 1.95, 2.11, 2.09, 1.98, 1.3]`

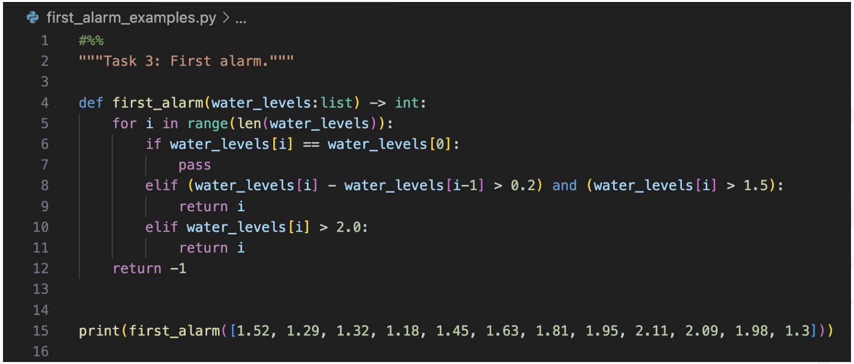
Special conditions


temp.py > ...

```python
def first_alarm(water_levels:list) -> int:
    for i in range(len(water_levels)):
        if (water_levels[i] - water_levels[i - 1] > 0.2 and water_levels[i] > 1.5):
            return i
            break
        elif water_levels[i] > 2:
            return i
            break

```

Mind your bounds!
Bonus question: When is the break reached?

Special conditions



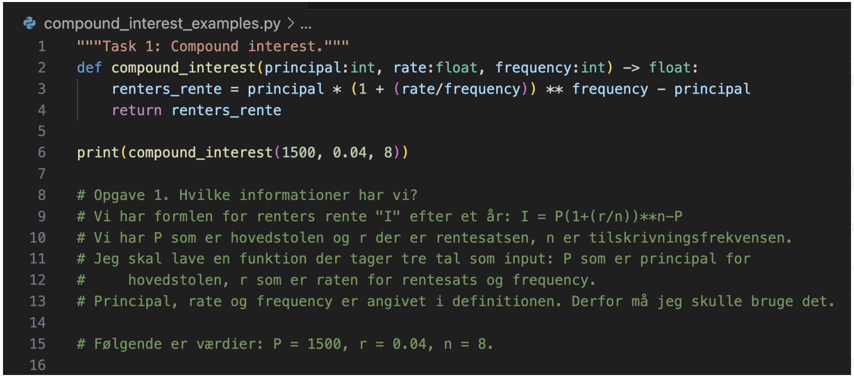
first_alarm_examples.py > ...

```python
#%%
"""Task 3: First alarm."""

def first_alarm(water_levels:list) -> int:
    for i in range(len(water_levels)):
        if water_levels[i] == water_levels[0]:
            pass
        elif (water_levels[i] - water_levels[i-1] > 0.2) and (water_levels[i] > 1.5):
            return i
        elif water_levels[i] > 2.0:
            return i
    return -1


print(first_alarm([1.52, 1.29, 1.32, 1.18, 1.45, 1.63, 1.81, 1.95, 2.11, 2.09, 1.98, 1.3]))

```

Bonus question: what about an empty list?

Only solution counts

We evaluate your code automatically.



compound_interest_examples.py > ...

```python
"""Task 1: Compound interest."""
def compound_interest(principal:int, rate:float, frequency:int) -> float:
    renters_rente = principal * (1 + (rate/frequency)) ** frequency - principal
    return renters_rente

print(compound_interest(1500, 0.04, 8))

# Opgave 1. Hvilke informationer har vi?
# Vi har formlen for renters rente "I" efter et år: I = P(1+(r/n))**n-P
# Vi har P som er hovedstolen og r der er rentesatsen, n er tilskrivningsfrekvensen.
# Jeg skal lave en funktion der tager tre tal som input: P som er principal for
#     hovedstolen, r som er raten for rentesats og frequency.
# Principal, rate og frequency er angivet i definitionen. Derfor må jeg skulle bruge det.

# Følgende er værdier: P = 1500, r = 0.04, n = 8.

```

No need for comments or messages.

Now:

- Check DE.
- Finish all tasks.
- Collect your code.
- Check your Python installation.
- Make no changes before the exam.
- (Extra practice: exams from earlier courses.)

Midterm solutions: Compound Interest

```python
def compound_interest(principal:int, rate:float, frequency:int) -> float:
    """Return the compound interest given principal, rate and frequency.

    :param principal: A positive integer, the principal sum.
    :param rate: A positive float, the interest rate.
    :param frequency: A positive integer, the compounding frequency.
    :return: The compound interest.
    """

    return (principal * (1 + rate / frequency) ** frequency) - principal

```

Midterm solutions: Stock Status

```python
def stock_status(number_of_items:int, days_to_delivery:int) -> str:
    """Return stock status message given number of items and days to delivery.

    :param number_of_items: An integer, the number of items in stock.
    :param days_to_delivery: An integer, the number of days to delivery.
    :return: The stock status message.
    """
    if number_of_items<0:
        return('Unknown')
    elif number_of_items>5:
        return('In stock')
    elif number_of_items>0:
        return('Only ' + str(number_of_items) + ' left in stock')
    else:
        if days_to_delivery<0:
            return('Unknown')
        elif days_to_delivery>0:
            return('Available in ' + str(days_to_delivery) + ' days')
        else:
            return('Out of stock')

```

Midterm solutions: First Alarm

```python
def first_alarm(water_levels:list) -> int:
    """Return the index of the first alarm given the list of water levels.

    :param water_levels: A list of floats, the water levels.
    :return: The index of the first alarm.
    """
    if water_levels[0] > 2.0:
        return 0
    for i in range(1, len(water_levels)):
        if water_levels[i] > 2.0 or (water_levels[i] > 1.5 and (water_levels[i] -
 water_levels[i-1]) > 0.2):
            return i
    return -1

```

Midterm solutions: Typical Successor

```python
def typical_successor(text:str, letter:str) -> str:
    """Return the letter that most often follows the given letter in the text.

    :param text: A string, the text.
    :param letter: A string, the letter.
    :return: The letter that most often follows the given letter.
    """

    text = text.lower()
    alphabet = 'abcdefghijklmnopqrstuvwxyz'

    my_dict = {}
    for l in alphabet:
        my_dict[l] = 0

    for i in range(len(text) - 1):
        if text[i] == letter and text[i + 1] in alphabet:
                my_dict[text[i + 1]] += 1

    current_max = 0
    successor = ''
    for key in my_dict:
        if my_dict[key] > current_max:
```

Midterm solutions: Dice Fariness

```python
def dice_fairness(throws:list) -> tuple:
    """Return the 3-element tuple containing dice statistics.

    :param throws: A list of integers, the throws of a dice.
    :return: A 3-element tuple containing information about the throws.
    """
    counts = [0, 0, 0, 0, 0, 0]
    for t in throws:
        counts[t - 1] += 1

    how_many = max(counts)
    most_frequent = counts.index(how_many) + 1
    expected = len(throws)/6

    return most_frequent, how_many, expected

```