Fundamentals of Computer Science - programming exam

Instructions

IMPORTANT - BEFORE YOU START:

1. Open and edit the `.py` file in Spyder to solve the exercises.
2. Upload the `.py` file to BBoard.

Additional notes

- The exam is **not open book**! No notes, smartphones, pen drives, ...
- **Clean your mess** after you finish! Comments and tests/asserts are welcome, but unnecessary code should be deleted.
- **Tests are provided** for each exercise, although they are initially commented out. They are very helpful to understand the intended behavior and to check the correctness of your implementation. Make sure to uncomment and run them as you go though the exercises..

Part 1 (40min)

Note: For the exercises in part 1, do not make use of libraries, python's built-in functions, sets, dictionaries, list's methods, etc.. unless otherwise stated. Just plain loops and if/else statements.

Ex. 1.1 - Procrustean String

Write a function called `procrustize` that takes two arguments as inputs, a string `s` and a non-negative integer `n`.

If the string `s` is longer than `n` , the function should truncate it to its first `n` characters. If it is shorter, it should pad it by appending extra spaces at the end, so that the resulting length is `n` . If the length is already `n` , it should leave it unchanged. It should return the resulting string.

For this exercise, you are allowed to use any form of indexing, string methods, and operators.

The function should check the argument types, and that `n` is non-negative, and raise an error if conditions are not met.

```
assert procrustize("hello", 4) == "hell"
assert procrustize("hello", 8) == "hello   "
```

Ex. 1.2 Periodic Fibonacci

Consider the Fibonacci sequence , a recursive sequence of numbers with the first two being 0 and 1, and the others being the sum of the previous two, that is $f_0 = 0$, $f_1 = 1$, $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$.

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, …
```

Now choose an integer $k \geq 2$, and take the remainder of the integer division of each term of the sequence with k.

For example, for `k=3`, you get:

```
0, 1, 1, 2, 0, 2, 2, 1, 0, 1, 1, 2, 0, 2, 2, 1, 0, 1, …
```

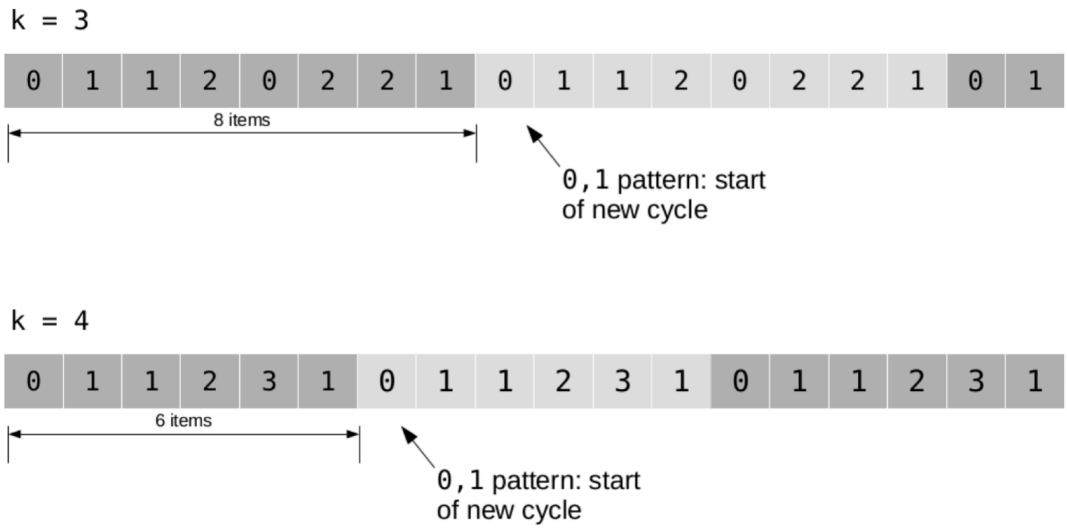
For `k=4` , you get:

```
0, 1, 1, 2, 3, 1, 0, 1, 1, 2, 3, 1, 0, 1, 1, 2, 3, 1, …
```

You can notice that the sequences are periodic, they repeat after a while. This actually happens for any k.

Your task is to write a function called `fibperiod` that takes an argument `k`, and determines the length of the period. For example, the period is `8` for `k=3` and it is `6` for `k=4`.

In order to detect the period, you should look for a place in the sequence where the two values `0` and `1` appear again in the sequence after the first time. See the examples in the picture below:



In the provided exam file you will already find a function that computes the Fibonacci sequence, it is called `fib` . Use it by calling it from within `fibperiod`.

```
assert fibperiod(3) == 8
assert fibperiod(4) == 6
```

Ex. 1.3 - Second Nearest

Define a function called `second_nearest` that takes as inputs a list `l` and a number `x` and returns th index and the value of the *second nearest* element to `x` (not the one closest to `x` in absolute distance, but the second closest). If the list has length 1 or less, return `(-1, 0)`. The result has to be determined using a single loop over the list.

```
res = second_nearest([4, 2.1, 1.8], 2)
assert res == (2, 1.8)

res = second_nearest([4, 2.1, 1.8], 1.9)
assert res == (1, 2.1)

res = second_nearest([4, 2.1, 1.8], 15)
assert res == (1, 2.1)

res = second_nearest([4, 2.1, 1.8], 3)
assert res == (0, 4)
```

Part 2 (40 min)

Exercise 2.1 - Sports League

The task in this exercise is to create a `SportsLeague` class which has a number of teams, each of which has a certain strength, that can organize matches between any two teams and a tournament.

Below is a list of what you have to do. No need to the check the type of the methods' inputs.

1. Write a class called `SportsLeague` whose constructor takes a single argument, a string denoting the league's name. The constructor should store this name in an internal attribute called `leaguename`, and also initialize another attribute called `teams` as an empty dictionary.

   ```
   In [1]: league = SportsLeague("The 43-Man Squamish National Association")
   In [2]: league.teams
   Out[2]: {}
   In [3]: league.leaguename
   Out[3]: 'The 43-Man Squamish National Association'
   ```

2. When the built-in function `len` is called on an object of this class, it should just return the length of the internal dictionary:

   ```
   In [1]: len(league)
   Out[1]: 0
   ```

3. Write a method called `addteam` that takes two arguments, a team `name` and its `strength`. It should then add the team in the internal dictionary:

```
In [5]: league.addteam("Shorts", 0.9)
In [6]: league.addteam("Talls", 1.1)
In [7]: league.addteam("Olds", 0.5)
In [8]: league.addteam("Mutants", 2.0)
In [9]: league.teams
Out[9]: {'Mutants': 2.0, 'Olds': 0.5, 'Shorts': 0.9, 'Talls': 1.1}
In [10]: len(league)
Out[10]: 4
```

4. Write a `strength` method that takes two arguments: a team `name` and `homeboost` , a `bool` that determines if the team is receiving a strength boost by playing at home. The returned value should be just the strength as stored in the internal dictionary if `homeboost` is `False` , but it should multiply it by 1.2 if `homeboost==True` :

```
In [1]: league.strength("Olds", False)
Out[1]: 0.5
In [2]: league.strength("Olds", True)
Out[2]: 0.6
```

5. Write a `match` method that takes the name of two teams as the arguments. Here is how it should work: it should use the `strength` method to get the teams strengths, giving the home boost to the first team. Then for each team it should compute a maximum number of points that it can make by computing a "relative strength" and multiplying it by 5: for example, for team 1, the formula to use is

$$m1 = int\left(5 \times \frac{s1}{s1 + s2}\right) \tag{2}$$

where `s1` and `s2` are the two strengths obtained above, and `int` denotes the integer part of the number (the floor operation). The formula for team 2 is analogous. You should then extract at random score (an integer number extracted with `random.randint`) for each team between 0 and its maximum (`m1` and `m2` respectively) and return the two scores. For example:

```
In [1]: league.match("Olds", "Mutants") # max scores are 1 and 3
Out[1]: (1, 0)
In [2]: league.match("Olds", "Mutants")
Out[2]: (1, 1)
In [3]: league.match("Olds", "Mutants")
Out[3]: (0, 2)
In [4]: league.match("Mutants", "Olds") # max scores are 4 and 0
Out[4]: (4, 0)
In [5]: league.match("Mutants", "Olds")
Out[5]: (3, 0)
```

6. Write a `tournament` method that takes no argument.

Here is how it should work: there should be two matches for each pair of teams in the league, where one of the two teams has the home advantage each time. In other words, each team should play twice against all the other teams, once at home and once not. For each match, you should determine from the scores if the first team has won, or the second one has won, or if the match was a draw. A win is worth 3 points, a draw is worth 1 point, a loss is worth 0 points. You should keep a dictionary that keeps track of the points that each team has made, and update it after each match.

The function should return the winning team (or any one of the winning teams in case of draw).

```
In [1]: league.tournament()
Out[2]: 'Mutants'
```

7. Modify the `tournament` method to also print the final scores in decreasing order as in the example below (*hint*: try with the `sorted` function, check its docstring).

```
In [1]: league.tournament()
Tournament results:
 Mutants 12
 Talls 11
 Shorts 6
 Olds 2
Out[2]: 'Mutants'
```