Computer Programming I: Exam

2020-10-23 08:00 - 13:00 + 30 minutes to hand in your solution on Studium.

Welcome to the examination for the course Computer Programming I. The exam consists of two parts, A and B. The first part contains a set of questions that require you to solve smaller problems, or answer questions about code. The second part is concerned with a slightly larger program, where you have to complete some parts that are omitted, while making sure that your solution works well with the provided code.

The solutions from previous questions can be built upon in the next questions, as long as not stated otherwise.
It is permitted (and sometimes recommended) to introduce new methods and functions. The statement "Write a function that" does not mean that your answer should not be structured with the help of extra functions.

All questions are related to the Python 3 programming language, and all your code should be written in this language. Extra care should be taken regarding readability (i.e. your code should be well structured and correctly indented). Variable, function, method and class names should be descriptive, but can usually be kept relatively short. Note that your grade can be negatively affected by, among other things:

- unnecessary variables,
- bad readability,
- repetition of identical snippets of code,
- failure to make good use of given snippets or code written earlier,
- badly ineffective code (for instance with many unnecessary function calls)

If you don't remember exactly what a function is called or how a part of the Python syntax looks like, you can point it out and describe which assumptions you make. We judge both how you solve the problem and how you can handle the Python programming language and it's functionalities.

Rules:
You **are** allowed to search Google, StackOverflow, Python documentation and similar sources on the Internet. If you base your solution on any such resource, give a link to it. You are further allowed to run your code on your computer and inspect the output (and use the debugger, extra test-cases).
You are under **no** circumstances allowed to communicate or work together with any other person (student or otherwise) during the exam. If you have any questions, contact the main teacher of the course, Johan Öfverstedt (johan.ofverstedt@it.uu.se). Plagiarism will not be tolerated and will result in disciplinary action including suspension from all studies at Uppsala University.

Grading
To pass this exam (grade 3), it is necessary that the A-part be mostly correct. This does not mean that every task needs to be exactly right, but you do need to demonstrate that you fulfill the course's goals.

For grade 4, it is necessary that at least half of the tasks of the B-part are completed and correct. You should solve all the tasks of the B-part to get grade 5. The code quality is also taken into consideration for these grades. Part B will not be graded unless part A is satisfactory completed.

Your solution containing all the answers should be submitted inside a single .txt or .py (if you make the code runnable) file to Studium. Mark each question with #A1, #A2, etc, to make searching within the document easy. Make sure to write your anonymous exam code at the top of the file.

Good luck!

Part A

A1. Describe in words (less than 100) what the following code does, and list all

- modules,
- function definitions,
- method definitions,
- function calls,
- method calls,
- classes,
- variables,
- attributes,
- constructors,
- parameters (not arguments).

```
import random

def fib(n):
    if n == 0:
         return 0
    if n == 1:
         return 1
    f0 = 0
    f1 = 1
    for _ in range(2, n):
        fi = f0 + f1
        f0 = f1
        f1 = fi
    return f1


class Dice:
    def __init__(self, nsides):
```

```
        self.nsides = nsides
        self.value = 0

    def roll(self):
        self.value = random.randint(1, self.nsides)

d = Dice(6)
d.roll()
print(fib(d.value))
```

A2. Write a function `alternating` that takes a string as parameter and returns a new string where every other character is uppercase and lowercase respectively.

Example:

```
print(alternating('This is all A part of the test.'))
# Should give the print-out: ThIs iS AlL A PaRt oF ThE TeSt.
```

A3. Write a function predecessors(s) that takes a string as parameter, splits it up into words (using the method `split`), transforms each word into lowercase (using the method `lower`), and returns a dictionary that contains, for each word, the words that preceded the word in the text.

Hint: You need to keep track of the previous word as you loop through the words of the string, and the previous word of the first word can be taken to be the empty string ''.

Example:

```
print(predecessors('Hello world the world is the best world in the
world'))
#{'hello': [''], 'world': ['hello', 'the', 'best', 'the'], 'the':
['world', 'is', 'in'], 'is': ['world'], 'best': ['the'], 'in':
['world']}
```

A4. Write a function central_difference(lst) that takes a list of numbers (floats or ints) and for each index in the list compute the central difference approximation g(x) = (f(x+1)-f(x-1)) / 2. The first and the last elements of the original list can be ignored; for a list of 7 elements as input, a list of 5 elements will be returned. The original list may not be altered within the function.

Example:

```
print('central_difference([-5, 7, 4, 9, 10, 11, 0]):',
central_difference([-5, 7, 4, 9, 10, 11, 0]))
# should give the output:
central_difference([-5, 7, 4, 9, 10, 11, 0]): [4.5, 1.0, 3.0, 1.0,
-5.0]
```

A5. Write a function `random_shuffle(lst)` that takes a list of elements and returns a new list as output which contains all the elements in the original list but reordered at random.

You may use functions from the `random` module, or `numpy.random`, but not use the `numpy.random.random_shuffle` function (or similar ready-made alternative) in your implementation. For the solution to be considered correct, it needs to be able to return different outputs when called with the same list (and succeed on the test which you write in the next question).

Example:

```
print(random_shuffle(['Hej', 5, [1, 2, 3], 'Python is great!']))
```
may give an output such as (but due to randomness, can give any other ordering):
```
[5, [1, 2, 3], 'Python is great!', 'Hej']
```

A6. Write a function test_random_shuffle(lst) that takes a list of elements as a parameter and aims to test the correctness of the previous function random_shuffle. The test needs to verify that:

1) the random_shuffle function does not modify the input list (possibly by making a copy, that can be compared against the list sent in as argument, after calling the function),
2) that all the elements in the original list are preserved (but potentially located at a different position in the list),
3) that the resulting list has the same length as the original list.

A7. Write a function `draw_text_circle(r, n)` that for a grid of 2*n+1 (positions -n to n) in each dimension prints a circle to the standard output (use a nested for-loop with printing using `end=''` to avoid line-breaks after each printed symbol). The magnitude of a given position is given by the formula `sqrt(x*x + y*y)` and we define being on the circle as having a position with a magnitude between r - 0.5 and r + 0.5.

The following example is printed when calling the function as `draw_text_circle(7, 9)`

```
..................
..................
.......XXXXX.......
.....XX.....XX.....
....X.........X....
...X...........X...
...X...........X...
..X.............X..
..X.............X..
..X.............X..
..X.............X..
..X.............X..
...X...........X...
...X...........X...
....X.........X....
.....XX.....XX.....
.......XXXXX.......
..................
..................
```

A8. We shall now create a class that represents a standard (digital) clock counting from 00:00:00 to 23:59:59. Write the class definition, using the name Clock, add a constructor that takes an initial time in the form of three parameters hour, minute, and second, and stores them as attributes. Then add a __str__ method that returns a string on the format 07:23:42 (including leading zeros).

Example:

```
# The following code:
c = Clock(7, 11, 13)
print(c)
# should give the following output:
07:11:13
```

A9. Add a method `tick` to the Clock class that takes a parameter `n` (with default value 1), that denotes how many seconds the clock should be incremented. Every time the second becomes 60, it should wrap around to 0 while incrementing the minute counter by 1, and when the minute counter becomes 60, it should increment the hour counter by 1. When the

hour counter becomes 24, it should wrap around to 0.

Example:

```
# The following code:
c = Clock()
c.tick(7*60*60)
print(c)
c.tick(72)
print(c)
c.tick(17*60*60)
print(c)
# should give the following output:
07:00:00
07:01:12
00:01:12
```

Part B

All the problems in the following set of problems relate to writing a solver of mazes.

You are provided the following code:

```
class Maze:
    def __init__(self, n):
        self.a = [['X' for _ in range(n)] for _ in range(n)]

    def create_path_between(self, x1, y1, x2, y2):
        xc = x1
        yc = y1
        self.a[yc][xc] = ' '
        while xc != x2 or yc != y2:
            if x2 > xc:
                xc += 1
            elif x2 < xc:
                xc -= 1
            if y2 > yc:
                yc += 1
            elif y2 < yc:
                yc -= 1
            self.a[yc][xc] = ' '

    def create_exit(self, x, y):
        self.a[y][x] = 'E'
```

```
    def get_neighbour(self, x, y):
        if x < 0 or y < 0 or x >= len(self.a) or y >= len(self.a):
            return (x, y, 'X')
        else:
            return (x, y, self.a[y][x])

    def get_neighbours(self, x, y): # *** B1: get_neighbours
        pass # replace pass with your code

    def __str__(self):
      s = ''
      s += 'X' * (len(self.a)+2)
      s += '\n'
      for y in range(len(self.a)):
          s += 'X'
          for x in range(len(self.a)):
              s += self.a[y][x]
          s += 'X\n'
      s += 'X' * (len(self.a)+2)
      return s

mz = Maze(10)
mz.create_path_between(0, 0, 9, 9)
mz.create_path_between(1, 1, 7, 3)
mz.create_path_between(7, 3, 5, 0)
mz.create_path_between(7, 3, 8, 5)
mz.create_path_between(8, 5, 7, 6)

mz.create_path_between(0, 0, 0, 4)
mz.create_path_between(0, 4, 4, 7)
mz.create_path_between(4, 7, 0, 9)
mz.create_path_between(0, 9, 0, 5)

mz.create_exit(9, 9)

print(mz)

mz_solver = RandomMazeSolver(0, 0, mz) # *** B2: RandomMazeSolver

print(mz)
while mz_solver.step() == False: # *** B3: step
    pass

pth = mz_solver.path
shrt_pth = shorten_path(pth) # *** B4: shorten_path

print('Long path: ', pth, sep='')
```

```
print('Short path: ', shrt_pth, sep='')
print(f'Length of long path: {len(pth)}.')
print(f'Length of short path: {len(shrt_pth)}.')
```

When run, the provided code gives the output

```
XXXXXXXXXXXX
X XXXX XXXXX
X  XXX XXXXX
X X XXX XXXX
X XX  XXX
X XXX XXX XX
X  XXX XX XX
X X XXX  XXX
X XX  XX XXX
X XX XXXX XX
X   XXXXXXEX
XXXXXXXXXXXX

Long path: [(0, 0), (0, 1), (1, 1), (0, 2), (0, 3), (0, 4), (0,
5), (0, 6), (0, 7), (0, 8), (0, 9), (0, 8), (0, 7), (0, 8), (0,
9), (0, 8), (0, 9), (1, 9), (0, 8), (0, 7), (0, 8), (0, 7), (0,
8), (0, 9), (1, 9), (2, 9), (3, 8), (2, 9), (1, 9), (0, 8), (0,
9), (0, 8), (0, 7), (0, 8), (0, 7), (0, 6), (0, 7), (0, 8), ...,
(7, 7), (8, 8), (7, 7), (8, 8), (9, 9)]
Short path: [(0, 0), (0, 1), (1, 1), (2, 2), (3, 3), (4, 4), (5,
5), (6, 6), (7, 7), (8, 8), (9, 9)]
Length of long path: 452.
Length of short path: 11.
```

Each space represents a walkable position in the maze and each X represents a wall in the maze, and the E represents the exit.

Now you will, using this provided code, write a program (consisting of functions and classes) which solves the maze by random walks (walking randomly between walkable positions until reaching the end point).

As a final step, the found path will be refined into a shorter path (but not necessarily the shortest possible).

B1. Write the body of the method `get_neighbours` that returns a list containing the 8 neighbours in the form of tuples `[(x, y, value), (x, y, value), ..., (x, y, value)]` on all sides of a cell given by coordinates x, y using the method `get_neighbour` which is provided already.

B2. Write the class definition of the class `RandomMazeSolver`, including the constructor, that represents a navigation agent trying to find the path to the exit, and that works with the

given code. (Note what is being passed in as arguments in the code where the solver object is created). Think about which attributes it will need. It should store the walked path as an attribute, it's current location, and a reference to the maze object.

B3. Write the method `step` in the class `RandomMazeSolver`, that takes a random step in the maze from the current location of the agent. Use the `get_neighbours` method of the maze class (B1) and use a `random_shuffle` (from part A) on the neighbour-list to facilitate randomness. Steps should only be taken to empty spaces and to the exit (not into walls). If the agent arrives at the exit after taking a step, the method returns `True`, otherwise `False`. Each new position that is visited should be added as a tuple (x, y) to the path attribute.

B4. Write the function `shorten_path` that takes a path (list of tuples (x, y)) and computes a shorter path from it by removing any detour (a part of the path that did not lead to any progress towards the exit). A detour can be defined as any subpath that starts from a given position and ends in that very same position.

Example: In the path [(1, 1), (2, 2), (2, 3), (2, 2), (3, 2), (3, 3)], the subpath [(2, 2), (2, 3), (2, 2)] was a detour, since if we walk [(1, 1), (2, 2), (3, 2), (3, 3)], we end up in the same position but with fewer steps.

The `shorten_path` function can be implemented in several ways. Here is one way that you can use for your implementation:

Description of a SHORTEN PATH algorithm
Initialize an index `i` to the index of the last element in the path-list.
Create a new empty path list.
Then do the following until the index `i` goes below 0:
  Find the first index `j` in the original path-list that contains the position which is found in the current (`i`:th) position `path[i]`. Add `path[i]` last in the new path list.
  Set `i` to be `j-1`.
Reverse the constructed path list (since you added the end first) and return the result.