Programming in Python
3rd Homework Solutions

Karlo Knežević
karloknezevic.github.io

November 15, 2021

1 Solutions

The solutions are intentionally given in *pdf* format with the aim of writing scripts by yourself.

1.1 Task 1

```
def factorial (n = 1):
        """
        Iterative version of computation
        factorial functions.
        """
        if n < 1:
                return 0

        fact = 1
        for i in range (1, n + 1):
                fact *= i

        return fact

def factorial_recursive (n):
        """
        Recursive version of the function
        factorial.
        """
        if n <1:
                return 0

        if n == 1:
                return 1

        #according to the definition: n! = n * (n-1)!
        return n * factorial_recursive (n-1)

if __name__ == "__main__":
        #correctivity testing
        n = 5
        print (f"{n}!␣is␣{factorial␣(n)}")
        print (f"{n}!␣is␣{factorial_recursive␣(n)}")
```

```
from ex1 import factorial, factorial_recursive as fr
```

```
if __name__ == ’__main__’:

        #entries until you enter a good value
        while True:
                n = input ("Enter␣a␣natural␣number:")

                if not n.isdigit ():
                        print ("You␣did␣not␣enter␣a␣natural␣number")
                        continue

                n = int (n)
                break

        print (f"{n}!␣=␣{factorial␣(n)}")
        print (f"{n}!␣=␣{fr␣(n)}")
```

1.2 Task 2

```
import math

def min_(l):
        if l == None or len(l) < 1:
                return "Empty␣list!"

        min_ = l[0]
        for e in l:
                if e < min_:
                        min_ = e

        return min_

def max_(l):
        if l == None or len(l) < 1:
                return "Empty␣list!"

        max_ = l[0]
        for e in l:
                if e > max_:
                        max_ = e

        return max_

def avg(l):
        if l == None or len(l) < 1:
                return "Empty␣list!"

        sum_ = 0
        for e in l:
                sum_ += e

        return sum_ / len(l)

def std(l):
        if l == None or len(l) < 2:
                print("Impossible␣to␣calculate␣std.")

        avg_ = avg(l)
        sum_squared = 0
        for e in l:
                sum_squared += (e - avg_)**2

        return math.sqrt(sum_squared/ (len(l)-1))

def med(l):
        l_srt = sorted(l)
        if len(l) % 2 != 0:
                median = l_srt[len(l) // 2]
        else:
                median = (l_srt[len(l) // 2] +
                        l_srt[len(l) // 2 + 1]) / 2

        return median

def med50(l):
        median = med(l)

        result = []
        threshold = 1.5 * median
        for e in l:
                if e > threshold:
                        result.append(e)

        return result

def list_print(l):
        if l == None or len(l) < 1:
                print("Empty␣list")
                return

        for e in l:
                print(e, end = "␣")
        print()
        return
```

```
import ex2
import random

def generate_list(n=10, min=-5, max=5):
        list_ = []
        for i in range(n):
                #generating values from a given interval
                value = min + random.random()*(max - min)
                list_.append(value)

        return list_

if __name__ == ’__main__’:


        #validation
        while True:
                n = int(input("Enter␣the␣number␣of␣list␣items:␣"))

                if n > 0:
                        break

        #validation
        while True:
                min_ = int(input("Enter␣the␣lower␣limit␣of␣the␣interval:
                   ␣"))
                max_ = int(input("Enter␣the␣upper␣limit␣of␣the␣interval:
                   ␣"))

                if max_ > min_:
                        break


        list_ = generate_list(n, min_, max_)

        ex2.list_print(list_)


        print(f"Min:␣{ex2.min_(list_):.2f}")
        print(f"Max:␣{ex2.max_(list_):.2f}")

        print(f"Avg:␣{ex2.avg(list_):.2f}")
        print(f"Std:␣{ex2.std(list_):.2f}")

        print(f"Median:␣{ex2.med(list_):.2f}")

        print(f"50%␣higher␣than␣median␣are␣{ex2.med50(list_)}")

        ex2.list_print(list_)
```

1.3 Task 3

```
def fibonacci_print(n):
        a = 0
        b = 1

        for i in range(n):
                print(f"{a}␣", end="")
                a, b = b, a+b
        print()
        return b

def fibonacci_find(n):
        """
        The first Fibonacci number is 0,
        and the other is 1.
        """
        a = 0
        b = 1

        for i in range(n-1):
                a, b = b, a+b
        return a

def fibonacci_recursive(n):
        """
        The first Fibonacci number is 0,
        and the other is 1.
        """
        if n == 1:
                return 0

        if n == 2:
                return 1

        #call counter
        print(f"{n:5d}", end = "\r")

        return fibonacci_recursive(n-1) + fibonacci_recursive(n-2)

if __name__ == ’__main__’:

        n = 30
        print(f"Fibo␣{n}␣is␣{fibonacci_find(n)}")
        print(f"Fibo␣{n}␣is␣{fibonacci_recursive(n)}")
        fibonacci_print(n)
```

```
import ex3

if __name__ == ’__main__’:

        #validation
        while True:
                n = input("Enter␣natural␣number␣n:␣")

                if not n.isdigit():
                        continue

                n = int(n)
                if n > 0:
                        break

        print(f"Iterative␣Fibo␣{n}␣is␣{ex3.fibonacci_find(n)}")
        print(f"Recursive␣Fibo␣{n}␣is␣{ex3.fibonacci_recursive(n)}")
        ex3.fibonacci_print(n)
```

1.4 Task 4

```
from math import sqrt

def is_prime(k):

        #with sqrt much faster solution!
        for i in range(2, int(sqrt(k)) + 1):
                if k % i == 0:
                        return False

        return True

def Mersenne_print(n):
        counter = 2
        while n > 0:

                #candidate for Mersenne number
                M = 2**counter - 1
                if is_prime(M):
                        n -= 1
                        print(f"M_{counter}␣=␣{M}")

                counter += 1
```

```
from ex4 import Mersenne_print

if __name__ == '__main__':

        n = int(input("Enter␣how␣many␣Mersenne␣numbers␣you␣want:␣"))

        Mersenne_print(n)
```

1.5 Task 5

```
def print_matrix(M=[[]], name = None):
        if name != None:
                print(name)

        for row in range(len(M)):
                for column in range(len(M[0])):
                        print(f"{M[row][column]:5.1f}", end="")
                print()
        print()

def add_matrices(M1, M2):
        s = [[0 for c in range(len(M1[0]))] for r in range(len(M1))]
        for row in range(len(M1)):
                for column in range(len(M1[0])):
                        s[row][column] = M1[row][column] + M2[row][
                           column]
        return s

def sub_matrices(M1, M2):
        s = [[0 for c in range(len(M1[0]))] for r in range(len(M1))]
        for row in range(len(M1)):
                for column in range(len(M1[0])):
                        s[row][column] = M1[row][column] - M2[row][
                           column]
        return s

def scalar_mul_matrix(scalar, M1):
        s = [[0 for c in range(len(M1[0]))] for r in range(len(M1))]
        for row in range(len(M1)):
                for column in range(len(M1[0])):
                        s[row][column] = scalar * M1[row][column]
        return s

if __name__ == '__main__':
        print("Functions␣can␣be␣tested␣here!")
```

```
import random
import ex5

if __name__ == '__main__':
        m = int(input("Enter␣M␣(number␣of␣rows)␣matrix␣dimension:␣"))
        n = int(input("Enter␣N␣(number␣of␣columns)␣matrix␣dimension:␣"))
        A = float(input("Enter␣a␣real␣number␣A:␣"))

        #list comprehension
        M1 = [[random.randint(0,10) for c in range(n)] for r in range(m)
           ]
        M2 = [[random.randint(0,10) for c in range(n)] for r in range(m)
           ]

        ex5.print_matrix(M1, "M1")
        ex5.print_matrix(M2, "M2")

        ex5.print_matrix(ex5.add_matrices(M1, M2), "M1␣+␣M2")
        ex5.print_matrix(ex5.sub_matrices(M1, M2), "M1␣-␣M2")
        ex5.print_matrix(ex5.scalar_mul_matrix(A, M2), "A␣*␣M1")
```

1.6 Task 6

```
import random

#generatorska funkcija jer sadrzi yield
def enumerate_my(itererable, counter = 0):
        for element in itererable:
                yield counter, element
                counter += 1

def zip_my(iterable1, iterable2):
        if len(iterable1) != len(iterable2):
                print("Collections␣are␣not␣the␣same␣length!")
                return

        return [(iterable1[i], iterable2[i]) for i in range(len(
           iterable1))]

def map_my(iterable, function):
        #na sve elemente primjeni predanu funkciju
        return [function(e) for e in iterable]

def filter_my(iterable, filter):
        #na sve elemente primjeni filtarsku funkciju
        return [e for e in iterable if filter(e)]


if __name__ == '__main__':
        N = 10

        #from 97 to 122 in the ascii table, but also utf8 are
        #chars a to z
        n1 = [chr(random.randint(97, 122))for i in range(N)]
        n2 = [chr(random.randint(97, 122))for i in range(N)]

        #generator
        for i, element in enumerate_my(n1):
                print(i, element, end = ",␣")
        print()

        print(zip_my(n1, n2))
        #convert all letters to uppercase
        print(map_my(n1, lambda slovo: slovo.upper()))

        #filter letters that have an even index in the acsii table
        print(filter_my(n1, lambda slovo: ord(slovo) % 2 == 0))

        def f1(letter):
                return letter + letter

        #filter all letters lower than k
        def f2(letter):
                return ord(letter) < ord("k")

        #make a duplicate
        print(map_my(n1, f1))

        print(filter_my(n1, f2))
```

1.7 Task 7

```
def pascal_triangle(n):
   trow = [1]
   y = [0]
   for x in range(max(n,0)):
      trow=[l+r for l, r in zip(trow+y, y+trow)]

   return trow

def x_plus_y(n):
        print(f"(x+y)^{n}␣=␣", end = "")
        koeff = pascal_triangle(n)

        for i in range(len(koeff)):
                if i > 0:
                        print("␣+␣", end = "")
                print(f"{koeff[i]}*x^{n-i}y^{i}", end = "")

if __name__ == '__main__':
        print(pascal_triangle(6))
        x_plus_y(6)
```

```
import ex7

if __name__ == '__main__':
        print(ex7.pascal_triangle(6))
        ex7.x_plus_y(6)
```

1.8 Task 8

```
from ex2 import min_, max_, avg, std, med

def load_file(path):
        """
        using width block
        """
        with open(path) as f:
                data = []
                for line in f:
                        line = line.strip()

                        #empty line
                        if len(line) == 0:
                                continue

                        parts = line.split(",")
                        #parse data
                        data.append(
                                (
                                        float(parts[0]),
                                        float(parts[1]),
                                        float(parts[2]),
                                        float(parts[3]),
                                        parts[4]
                                )
                        )
        return data

def return_list(index, data):
        """
        Return data list.
        """
        return [d[index] for d in data]

def print_both(content, f = None):
        """
        Print to standart output or to a file.
        """
        print(content)

        if f != None:
                f.write(f"{content}\n")

def calculate_stat(i, data, f):
        """
        Min, max, avg, std and med.
        """
        print_both(f"Min(attribute_{i})␣=␣{min_(return_list(i,␣data))}",
           f)
        print_both(f"Max(attribute_{i})␣=␣{max_(return_list(i,␣data))}",
           f)
        print_both(f"Avg(attribute_{i})␣=␣{avg(return_list(i,␣data)):.2f
           }",f)
        print_both(f"Std(attribute_{i})␣=␣{std(return_list(i,␣data)):.2f
           }",f)
        print_both(f"Med(attribute_{i})␣=␣{med(return_list(i,␣data))}",f
           )
        print_both(20*"-",f)


if __name__ == '__main__':

        data = load_file("../data/iris/iris.data")
        f = open("iris.txt", "w")

        print_both(data, f)
        #print stat
        for i in range(4):
                calculate_stat(i, data, f)

        #reach flower classes
        #use a set for uniques
        razred = set(return_list(4, data))
        print(razred)

        #r are the names of the classes
        for r in razred:
                #retrieve all those data whose class is equal to r
                data_r = list(filter(lambda t: t[4] == r, data))
                print_both(f"Class␣statistics:{r}", f)
                calculate_stat(i, data_r, f)

        f.close()
```

1.9 Task 9

```
from ex2 import min_, max_, avg, std, med
import csv

#global constants
PRICE = 25 #float
NAME = 2
HORSEPOWER = 21 #float
FUEL_TYPE = 3
CYLINDERS = 15

index = [PRICE, NAME, HORSEPOWER, FUEL_TYPE, CYLINDERS]

def load_file(path):
        with open(path, newline="") as f:
                reader = csv.reader(f, delimiter=",")
                data = []
                for line in reader:

                        #parse
                        for i in index:
                                if i == PRICE or i == HORSEPOWER:
                                        line[i] = float(line[i]) if line
                                            [i] != "?" else 0

                        data.append((line))
        return data

def lowest_price(data):
        najmanja_cijena = min_([e[PRICE] for e in data if e[PRICE] > 0])

        najjefitniji = [d for d in data if d[PRICE] == najmanja_cijena]

        return najjefitniji

def highest_price(data):
        najveca_cijena = max_([e[PRICE] for e in data])
        najskuplji = [d for d in data if d[PRICE] == najveca_cijena]

        return najskuplji

def avg_price(data, tip):
        return avg([d[PRICE] for d in data if d[NAME] == tip])

def max_horsepower(data):
        max_konji = max_([e[HORSEPOWER] for e in data])

        najjaci = [d for d in data if d[HORSEPOWER] == max_konji]

        return najjaci

def gasoline_share(data):
        benzinci = sum([1 for d in data if d[FUEL_TYPE] == "gas"])

        return benzinci/len(data)

def max_cylinders(data):
        c = ["two","three","four","five","six","eight","twelve"]
        c.reverse()

        for cil in c:
                r = []
                for d in data:
                        if cil == d[CYLINDERS]:
                                r.append(d)

                if len(r) > 0:
                        break

        return r

def print_cars(data):
        imena = set([d[NAME] for d in data])

        #sortiraj po cijeni
        data.sort(key = lambda d:d[PRICE])

        for i in imena:
                with open(f"auti/{i}.txt", "w") as f:
                        for d in data:
                                if d[NAME] == i:
                                        f.write(str(d) + "\n")

if __name__ == ’__main__’:

        data = load_file("../data/automobili/imports-85.data")
        #print(data)


        print(f"The␣cheapest␣cars␣are␣{lowest_price(data)}")

        print(f"The␣most␣expensive␣cars␣are␣{highest_price(data)}")

        print(f"Audi␣costs␣on␣average␣{avg_price(data,␣’audi’):.2f}")
```

```
    print(f"Cars␣have␣the␣most␣horesepowers␣are␣{max_horsepower(data
       )}")

    print(f"The␣share␣of␣gasoline␣cars␣is␣{gasoline_share(data)
       *100:.2f}")

    print(f"The␣car␣with␣the␣most␣cylinders␣is␣{max_cylinders(data)}
       ")

    print_cars(data)
```