Ekaterina Smirnova

[Название организации]

[Адрес организации]

PYTHON ASSIGNMENTS Examples

Here are code that has been written by me with explanations (The most valuable examples, to my opinion). The rest is available on GitHub

1.Sudoku Grid. The exercise shows knowledge in lists within lists.

function sudoku\_grid\_correct(sudoku: list), which takes a two-dimensional array representing a sudoku grid as its argument. The function should use the functions from the three previous exercises to determine whether the complete sudoku grid is filled in correctly. The function should check each of the nine rows, columns and 3 by 3 blocks in the grid. If all contain each of the numbers 1 to 9 at most once, the function returns True. If a single one is filled in incorrectly, the function returns False.

Code:

def row\_correct(sudoku: list, row\_no:int):

for i in range(len(sudoku[row\_no])):

count = sudoku[row\_no].count(i+1)

if count > 1:

return False

return True

def column\_correct(sudoku: list, column\_no: int):

stol = []

for i in range(len(sudoku)):

if sudoku[i][column\_no] in stol and sudoku[i][column\_no] != 0:

return False

else:

stol.append(sudoku[i][column\_no])

return True

def block\_correct(sudoku: list, row\_no: int, column\_no: int):

mas=[]

for row\_i in range(3):

for column\_i in range(3):

val = sudoku[row\_no+row\_i][column\_no+column\_i]

if val in mas and val != 0:

return False

else:

mas.append(val)

return True

def sudoku\_grid\_correct(sudoku:list):

for i in range(9):

f1 = row\_correct(sudoku, i)

f2 = column\_correct(sudoku, i)

if not f1 or not f2:

return False

for row in range(3):

for col in range(3):

f3 = block\_correct (sudoku, row\*3, col\*3)

if not f3:

return False

return True

if \_\_name\_\_ == "\_\_main\_\_":

sudoku1 = [

[9, 0, 0, 0, 8, 0, 3, 0, 0],

[2, 0, 0, 2, 5, 0, 7, 0, 0],

[0, 2, 0, 3, 0, 0, 0, 0, 4],

[2, 9, 4, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 7, 3, 0, 5, 6, 0],

[7, 0, 5, 0, 6, 0, 4, 0, 0],

[0, 0, 7, 8, 0, 3, 9, 0, 0],

[0, 0, 1, 0, 0, 0, 0, 0, 3],

[3, 0, 0, 0, 0, 0, 0, 0, 2]

]

print(sudoku\_grid\_correct(sudoku1))

sudoku2 = [

[2, 6, 7, 8, 3, 9, 5, 0, 4],

[9, 0, 3, 5, 1, 0, 6, 0, 0],

[0, 5, 1, 6, 0, 0, 8, 3, 9],

[5, 1, 9, 0, 4, 6, 3, 2, 8],

[8, 0, 2, 1, 0, 5, 7, 0, 6],

[6, 7, 4, 3, 2, 0, 0, 0, 5],

[0, 0, 0, 4, 5, 7, 2, 6, 3],

[3, 2, 0, 0, 8, 0, 0, 5, 7],

[7, 4, 5, 0, 0, 3, 9, 0, 1]

]

print(sudoku\_grid\_correct(sudoku2))

2.Creating Movie Database

The function named add\_movie(database: list, name: str, director: str, year: int, runtime: int), which adds a new movie object into a movie database.

The database is a list, and each movie object in the list is a dictionary. The dictionary should contain the following keys.

* name
* director
* year
* runtime

Code:

def add\_movie(database: list, name: str, director: str, year: int, runtime: int):

database.append({"name":name,"director":director,"year":year,"runtime":runtime})

if \_\_name\_\_=="\_\_main\_\_":

database = []

add\_movie(database, "Gone with the Python", "Victor Pything", 2017, 116)

add\_movie(database, "Pythons on a Plane", "Renny Pytholin", 2001, 94)

print(database)

3.Find Movies.

The exercise shows expertise using dictionaries.

The function named find\_movies(database: list, search\_term: str), which processes the movie database created in the previous exercise. The function should formulate a new list, which contains only the movies whose title includes the word searched for. Capitalisation is irrelevant here. A search for ana should return a list containing both Anaconda and Management.

Code:

def find\_movies(database: list, search\_term: str):

x=[]

for movie in database:

if search\_term in movie["name"].lower():

x.append(movie)

return x

if \_\_name\_\_=="\_\_main\_\_":

database = [{"name": "Gone with the Python", "director": "Victor Pything", "year": 2017, "runtime": 116},

{"name": "Pythons on a Plane", "director": "Renny Pytholin", "year": 2001, "runtime": 94},

{"name": "Dawn of the Dead Programmers", "director": "M. Night Python", "year": 2011, "runtime": 101}]

my\_movies = find\_movies(database, "python")

print(my\_movies)

4. Student database

Firstly, we create a function to add students and print out information of each one. Then, we add completed by each student course. The course info is tuple consisting of the course name and grade. Courses with grade=0 must be ignored and if the course is repeated by a single student, the better grade should be counted in. Summary prints out all the information stored in database.

Code: def add\_student(students:dict, name):

students[name] = {}

def print\_student(students:dict, name):

if name in students:#check wheather the name is in dictionary and if so we print the name

print(f"{name}:")

if "courses" in students[name]:# we check wether the student has courses at all

print(f" {len(students[name]['courses'])} completed courses:")# we counted how many courses the student has completed

for course in students[name]["courses"]:#creating a loop for go over the courses to print them out

print(f" {course[0]} {course[1]}") #where course[0] is a name of the course and course[1] is a grade

print(f" average grade {students[name]['av\_grade']}")# Print average grade

else:

print(" no completed courses")

if name not in students:# Creating a special condition for students who are not in db

print(name+": no such person in the database")

def add\_course(students:dict, name:str, course:tuple):# Creating a function to add courses

if course[1]>0: #We do not need course with grade lower then 1

if "courses" in students[name]: # Checking whether student has any courses

for c in range(len(students[name]["courses"])): # go over tuple

if students[name]["courses"][c][0]==course[0]: #Whether we have course with the same name

if course[1] > students[name]["courses"][c][1]: # if so we break the loop

students[name]["courses"][c] = course

else:

break

else: #if loop is not broken (aka'we didnt find the same course'),we continue

if course not in students[name]["courses"]:

students[name]["courses"].append(course)#unique courses are added to list new el

sum\_av\_grade=0 #creating var for counting av grade

for course in students[name]["courses"]: # go over each tuple for picking out grades for courses to calculate average grade

sum\_av\_grade+=course[1] #counter helps us to calculate all them together

students[name]["av\_grade"]=sum\_av\_grade/len(students[name]["courses"])# writing down calculated average grade

else:# if the course has not been added yet

students[name]["courses"] = [course] #creating key ['courses] with list value consisting course that was passed

students[name]["av\_grade"] = float(course[1]) #creating key ['av\_grade'] for calculating av grade for those students who do not have other courses

def summary(students:dict):

print(f"students {len(students)}")# for counting keys, use just len function as for lists

a=0

b=" "

for name in students:

if len(students[name]["courses"])>a:

a=len(students[name]["courses"])

b=name

print(f'most courses completed {a} {b}')

a=1

b=" "

for name in students:

if students[name]["av\_grade"]>a:

a=students[name]["av\_grade"]

b=name

print(f'best average grade {a} {b}')

if \_\_name\_\_ == "\_\_main\_\_":

students = {}

add\_student(students, "Peter")

add\_course(students, "Peter", ("Software Development Methods", 5))

summary(students)

5. Reading files/Recipe search:

This exercise is about creating a program which allows the user to search for recipes based on their names, preparation times, or ingredients used. The program should read the recipes from a file submitted by the user.

Code:

def get\_recipes(filename):

a = []

with open(filename) as new\_file:

st = ""

for line in new\_file:

if line != "\n":

st += line

else:

a.append(st[:-1])

st=""

a.append(st[:-1])

recipes = []

for i in a:

micro\_a = i.split("\n")

recipes.append({

"name": micro\_a[0],

"time": int(micro\_a[1]),

"ingredients": micro\_a[2:]

})

return recipes

def search\_by\_name(filename: str, word: str):

recipes = get\_recipes(filename)

otvet = []

for i in recipes:

if word in i["name"].lower():

otvet.append(i["name"])

return otvet

def search\_by\_time(filename: str, prep\_time: int):

recipes = get\_recipes(filename)

otvet = []

for i in recipes:

if prep\_time > i["time"]:

otvet.append(f'{i["name"]}, preparation time {i["time"]} min')

return otvet

def search\_by\_ingredient(filename: str, ingredient: str):

recipes = get\_recipes(filename)

otvet = []

for i in recipes:

if ingredient in i["ingredients"]:

otvet.append(f'{i["name"]}, preparation time {i["time"]} min')

return otvet

if \_\_name\_\_=="\_\_main\_\_":

found\_recipes = search\_by\_ingredient("recipes1.txt", "eggs")

for recipe in found\_recipes:

print(recipe)

6.Writing files/Find words

The exercise template includes the file words.txt, which contains words in English.

The function named find\_words(search\_term: str) returns a list containing all the words in the file which match the search term.

The search term may include lowercase letters and the following wildcard characters:

* A dot . means that any single character is acceptable in its place. For example, ca. would yield words like *cat* and *car*, p.ng would yield words like *ping* and *pong*, and .a.e would yield words like *sane*, *care* and *late*.
* An asterisk \* at the *end* of the search term means that any word which *begins* with the search term is acceptable. An asterisk at the *beginning* of the search term means that any word which *ends* with the search term is acceptable. For example, ca\* would yield words like *california*, *cat*, *caring* and *catapult*, while \*ane would yield words like *crane*, *insane* and *aeroplane*. There can only ever be a single asterisk in the search term.
* If there are no wildcard characters in the search term, only words which match the search term exactly are returned.

Code:

def find\_words(search\_term: str):

with open("words.txt","r") as my\_dict:

word\_list=[]

if "\*" in search\_term:

for word in my\_dict:

word=word.strip() #removes \n

if word.endswith(search\_term[1:]) or word.startswith(search\_term[:-1]):

word\_list.append(word)

elif "." in search\_term:

for word in my\_dict:

word=word.strip()

if len(search\_term)==len(word):

for i in range(len(word)):

if search\_term[i] != word[i] and search\_term[i] != ".":

break

else:

word\_list.append(word)

else:

for word in my\_dict:

word=word.strip()

if len(search\_term)==len(word):

for i in range(len(word)):

if search\_term[i] != word[i]:

break

else:

word\_list.append(word)

return word\_list

if \_\_name\_\_ == "\_\_main\_\_":

print(find\_words("\*vokes"))

print(find\_words("cat"))

7. Handling errors

The file lottery\_numbers.csv containts winning lottery numbers. Each line should contain a header week x, followed by seven integer numbers which are all between 1 and 39 inclusive. function named filter\_incorrect(), which creates a file called correct\_numbers.csv. The file should contain only those lines from the original file which are in the correct format.

Code:

def filter\_incorrect():

all\_correct=""

with open("lottery\_numbers.csv","r") as lottery\_num:

for line in lottery\_num:

try:

new\_line=line.split(" ")

new\_line=new\_line[1].split(";")

if int(new\_line[0])>0:

nums = new\_line[1][:-1].split(",")

if len(nums) == 7 and len(nums) == len(set(nums)):

for n in nums:

if int(n) > 0 and int(n)<=39:

pass

else:

raise ValueError()

all\_correct+=line

except:

pass

with open("correct\_numbers.csv","w") as correct\_num:

correct\_num.write(all\_correct)

if \_\_name\_\_ == "\_\_main\_\_":

filter\_incorrect()

8.Randomness/ Password Generator.

The function now takes three arguments:

* If the second argument is True, the generated password should also contain one or more numbers.
* If the third argument is True, the generated password should also contain one or more of these special characters: !?=+-()#.

Despite these two additional arguments, the password should always contain at least one lowercase alphabet.

Code: import random

import string

def generate\_strong\_password(number:int, p1:bool, p2:bool):

random\_chars=[]

numbers="0123456789"

punctuation= "!?=+-()#"

for i in range(number):

random\_char=random.choice(string.ascii\_lowercase)

random\_chars.append(random\_char)

if p1==True:

for i in range(random.randint(1, number//3)):

random\_chars[random.randint(1, number//2)] = random.choice(numbers)

if p2==True:

for i in range(random.randint(1, number//3)):

random\_chars[random.randint((number//2)+1, number-1)]= random.choice(punctuation)

random.shuffle(random\_chars)

return "".join(random\_chars)

if \_\_name\_\_ == "\_\_main\_\_":

for i in range(10):

print(generate\_strong\_password(8, True, True))