Федеральное государственное автономное образовательное учреждение высшего образования

«ОМСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Прикладная математика и фундаментальная информатика»

Практическое занятие №8

по дисциплине «Практикум по программированию»

на тему: «Функции в Python. Работа с файлами»

Вариант №14

Выполнил

Студент гр. **ФИТ-212** **Курпенов К.И.**

группа Фамилия И,О. студента подпись

Принял:

Преподаватель **Моисеева Н.А.**

Фамилия И.О, преподавателя дата, подпись

Омск 2022

ЗАДАНИЕ 1.

Условие:

Представим следующую ситуацию: вы являетесь сотрудником IT отдела некой компании. Ваша компания открывает новый филиал с переводом необходимого количества сотрудников из разных городов. Новый филиал имеет свой почтовый домен. Вам, как сотруднику IT отдела, необходимо создать каждому сотруднику, переходящему в новый филиал, свой уникальный почтовый адрес. У вас есть неотформатированный список сотрудников, который заполняли непонятно как. Также у вас есть кусок кода - функция, которая по имени и фамилии сотрудника составляет его уникальный почтовый адрес. Для выполнения данного задания вам необходимо написать программу, удовлетворяющую следующим требованиям:

1) Программа должна читать исходный текстовый файл

2) Программа должна содержать функцию, представленную ниже, для создания почтовых адресов

3) Программа должна заполнить пустой столбец с почтовыми адресами в исходном файле и перезаписать его

4) Почтовый адрес должен создаваться только для тех, у кого заполнены все остальные поля - Имя, Фамилия, Телефонный номер, Город

5) Телефонный номер считается валидным, если он состоит из 7 цифр, в противном случае информация невалидна и программа не должна создавать почтовый адрес сотруднику

После выполнения вашей программы исходный текстовый файл должен быть заполнен информацией с почтовыми адресами сотрудников.

Решение:

```python
import csv


def reader(path: str, encoding="utf-8") -> list:
    data = []

    with open(path, encoding=encoding) as file:
        buffer = csv.reader(file)
        for row in buffer:
            data.append(row)

    for i in range(len(data)):
        for j in range(len(data[i])):
            data[i][j] = data[i][j].strip()

    return data


def generate_emails(names: list) -> list:
    emails = []

    for name in names:
        email = f"{name[0]}.{name[1]}@company.io"
        letter = 1

        while email in email:
            email = f"{name[0]}.{name[1][:letter]}@company.io"
            letter += 1

        emails.append(email)

    return emails


```

```python
def writer(data: list, path: str, encoding="utf-8") -> None:
    old_data = []

    with open(path, "r", encoding=encoding) as file:
        buffer = csv.reader(file)
        for row in buffer:
            old_data.append(row)

    for i in range(len(old_data)):
        for j in range(len(old_data[i])):
            old_data[i][j] = old_data[i][j].strip()

    if len(data) != len(old_data) - 1:
        return

    new_data = []
    for i in range(len(old_data)):
        new_data.append(old_data[i])
        new_data[i][0] = data[i]

    with open(path, "w", encoding=encoding) as file:
        buffer = csv.writer(file)
        buffer.writerows(new_data)


if __name__ == "__main__":
    data = reader("task_file.txt")

    names = []
    for i in range(len(data)):
        names.append([data[1], data[2]])

    emails = generate_emails(names)

    writer(emails, "task_file.txt")
```

ЗАДАНИЕ 2.

Задача 2.1.

Условие:

Напишите функцию `horse()` для определения правильности хода коня в шахматах. Аргументы функции – две строки: положение коня на доске и клетка, в которую конь хочет походить. Если ход возможен, вывести `True`, иначе `False`.

Решение:

```python
def horse(start: str, stop: str) -> bool:
    words = "abcdefgh"

    symbol = words.find(start[0])
    digit = int(start[1])

    motions = []

    if 0 < symbol - 2 and digit + 1 <= 8:
        motions.append(f"{words[symbol - 2]}{digit + 1}")
    if 0 < symbol - 1 and digit + 2 <= 8:
        motions.append(f"{words[symbol - 1]}{digit + 2}")
    if symbol + 1 <= 8 and digit + 2 <= 8:
        motions.append(f"{words[symbol + 1]}{digit + 2}")
    if symbol + 2 <= 8 and digit + 1 <= 8:
        motions.append(f"{words[symbol + 2]}{digit + 1}")
    if symbol + 2 <= 8 and 0 < digit - 1:
        motions.append(f"{words[symbol + 2]}{digit - 1}")
    if symbol + 2 <= 8 and 0 < digit - 2:
        motions.append(f"{words[symbol + 2]}{digit - 2}")
    if 0 < symbol - 1 and 0 < digit - 2:
        motions.append(f"{words[symbol - 1]}{digit - 2}")
    if 0 < symbol - 2 and 0 < digit - 1:
        motions.append(f"{words[symbol - 2]}{digit - 1}")

    return stop in motions


if __name__ == "__main__":
    print(horse("d4", "b3"))
    print(horse("d4", "a3"))
```

Задача 2.2.

Условие:

Напишите функцию `brackets()` для проверки правильности расстановки скобок четырех видов: `[]`, `()`, `{}`, <>. Функция возвращает `True`, если скобки расставлены правильно, и `False`, если неправильно.

Решение:

```python
def brackets(sequence: str) -> bool:
    brackets_dict = {"]": "[", ")": "(", "}": "{", ">": "<"}

    stack = []

    for symbol in sequence:
        if symbol in "[({<":
            stack.append(symbol)
        elif symbol in "])}>":
            if brackets_dict[symbol] != stack.pop():
                return False
    return True


if __name__ == "__main__":
    print(brackets("[12 / (9) + 2(5{15 * <2 - 3>}6)]"))
    print(brackets("1{2 + [3}45 - 6] * (7 - 8)9"))
```

Задача 2.3.

Условие:

Напишите функцию `diversity()`, которая при каждом вызове возвращает, сколько раз она уже вызывалась именно с этим аргументом.

Решение:

```python
def diversity(arg: str) -> int:
    global calls
    calls.append(arg)
    return calls.count(arg)


if __name__ == "__main__":
    calls = []

    print(diversity("Happy"))
    print(diversity("New"))
    print(diversity("Year"))
    print(diversity("Year"))
    print(diversity("Year"))
```

Задача 2.4.

Условие:

Представьте, что вам нужно много раз находить количество делителей числа. Но эти числа могут повторяться. Чтобы не делать одну и ту же работу несколько раз, вы записали во внешнюю переменную `divisors` словарь: ключ — число, значение — количество делителей, и просто проверяете, есть ли такое число в словаре. Напишите функцию `number_of_divisors(n)` для поиска количества делителей числа `n`.

Решение:

```python
def number_of_divisors(divisor: int) -> int:
    global requests
    global divisors

    if divisor in divisors:
        return divisors[divisor]

    requests.add(divisor)

    count = 2

    for i in range(2, int(divisor**0.5) + 1):
        if not (divisor % i):
            count += 2

    divisors[divisor] = count

    return count


if __name__ == "__main__":
    requests = set()
    divisors = {}

    print(number_of_divisors(2000000000))
    print(number_of_divisors(6))
    print(number_of_divisors(6))
    print(number_of_divisors(2000000000))

    print(divisors)
```

Задача 2.5.

Условие:

Решите задачу, обратную нахождению степени числа, то есть по числу и основанию степени определите показатель степени. Напишите функцию `degree_indicator()`, принимающую число и основание степени. Функция возвращает показатель степени, в которую нужно возвести основание, чтобы получить число. Число точно является степенью основания, проверять это не нужно.

Решение:

```python
def degree_indicator(number: float, base: int) -> int:
    count = 0

    if number == 1:
        return 0
    elif number == base:
        return 1
    elif number < 1:
        while number != base**count:
            count -= 1
    elif number > 1:
        while number != base**count:
            count += 1
    return count


def func(number: int) -> int:
    if number == 0:
        return 0
    elif not (number % 3):
        return func(number // 3) + 1
    else:
        return 2 + func(number - 1)


if __name__ == "__main__":
    print(degree_indicator(81, 3))
    print(degree_indicator(1 / 625, 5))
    print(func(6))
    print(func(10000))
```

Задача 2.6.

Условие:

Напишите функцию `quarters()` для подсчета точек, находящихся в каждой из четвертей координатной плоскости. Функции передается произвольное количество кортежей координат точек, а возвращает она словарь, в котором записано, сколько точек находится в соответствующей четверти. Точки на осях координат в подсчете не участвуют.

Решение:

```python
def quarters(data: list) -> tuple:
    output = []

    for point in data:
        if point[0] > 0 and point[1] > 0:
            output.append(1)
        elif point[0] > 0 and point[1] < 0:
            output.append(4)
        elif point[0] < 0 and point[1] > 0:
            output.append(2)
        elif point[0] < 0 and point[1] < 0:
            output.append(3)

    return output.count(1), output.count(2),\
            output.count(3), output.count(4)


if __name__ == "__main__":
    print(quarters([(1, 1), (-1, 2), (-3, -1)]))
    print(quarters([(-5, 1), (-1, 3), (2, -1), (0, 3)]))
```

Задача 2.7.

Условие:

На брезентовом прямоугольном поле растут только вечнозеленые помидоры и алюминиевые огурцы. Считается, что это помидорное поле, если в каждом ряду помидоров не меньше, чем огурцов. На грядке огурцы – это любые числа, содержащие ноль, помидоры – все остальные. Определите, можно ли считать поле помидорным (EVERGREEN TOMATOES). Или огурцовым (ALUMINUM CUCUMBERS). И выведите свой вердикт.

Решение:

```python
def status(data: list) -> str:
    lines = []

    for line in data:
        c = 0
        t = 0
        for vegetable in line:
            if "0" in str(vegetable):
                c += 1
            else:
                t += 1
        lines.append(1 if c <= t else 0)

    if all(lines):
        return "EVERGREEN TOMATOES"
    return "ALUMINIUM CUCUMBERS"


if __name__ == "__main__":
    print(status([
            [1, 3, 6, 20],
            [2, 2, 10, 70, 50],
            [20, 10, 1]
        ]))

    print(status([
            [6, 20, 7, 103],
            [17],
            [1, 402, 14],
            [18, 3, 201, 909]
        ]))
```