1. **Создадим приложения, с которого начинается изучение всех классических языков**

**программирования. На первом этапе выведите в консоль приветствие Hello, world!**

print("Hello, world!")

1. **Создайте переменную greeting и задайте ей значение Hello, world!**  
     
   greeting = "Hello, world!"
2. **Выведите значение переменной greeting. Поздравляем! Вы создали свое первое**

**приложение в python**  
  
print(greeting)

1. **Необходимо расчитать, сколько плитки понадобится для проведения ремонта в комнате! Создать переменные length и width и присвоить в них значения 8 и 10 соответственно.**

length = 8

width = 10

tile\_count = length \* width # количество плитки для текущих размеров

1. **Оказалось, что произошла ошибка в расчетах и в длину необходимо 20 плиток. Какое количество плитки понадобится в этом случае?**

new\_length = 20

new\_tile\_count = new\_length \* width # количество плитки при новой длине

new\_tile\_count

1. **Самостоятельно! Рассчитать площадь прямоугольника со сторонами 23, 13**

length\_rect = 23

width\_rect = 13

rectangle\_area = length\_rect \* width\_rect

1. **Определите, являются ли следующие логические выражения истинными или ложными.**

Утверждение первое:

(6 \* 6) - 1 == 8 + 1

statement\_1 = (6 \* 6) - 1 == 8 + 1

print("Утверждение первое:", statement\_1)

Утверждение второе:

13 - 7 != (3 \* 2) + 1

statement\_2 = 13 - 7 != (3 \* 2) + 1

print("Утверждение второе:", statement\_2)

Утверждение третье:

3 \* (2 - 1) == 4 - 1

statement\_3 = 3 \* (2 - 1) == 4 - 1

print("Утверждение третье:", statement\_3)

1. **Определите, являются ли следующие логические выражения истинными или ложными.**

Утверждение первое:

(6 \* 6) - 1 >= 8 + 1

statement\_1 = (6 \* 6) - 1 >= 8 + 1

print("Утверждение первое:", statement\_1)

Утверждение второе:

13 - 7 <= (3 \* 2) + 1

statement\_2 = 13 - 7 <= (3 \* 2) + 1

print("Утверждение второе:", statement\_2)

Утверждение третье:

3 \* (2 - 1) > 4 - 1

statement\_3 = 3 \* (2 - 1) > 4 - 1

print("Утверждение третье:", statement\_3)

**Задание:**

1. Создайте переменную с именем bool\_variable и установите для нее значение true. Попробуйте вывести ее в консоль. Какая ошибка у вас вывелась? Почему?

2. Замените значение в bool\_variable на ‘true’ (в кавычках) Проверьте тип bool\_variable, используя функцию проверки типов type (bool\_variable).

Выведите значение в консоль. Какой тип теперь у переменной bool\_variable?

3. Почему это не логическая переменная!

Создайте переменную с именем bool\_variable\_2 и сделайте так, чтобы она имелв логический тип истины.

4. Проверьте тип bool\_variable\_2 и убедитесь, что вы успешно создали логическую переменную.

**Код:**

bool\_variable = 'true' # Теперь это строка

print("Значение переменной bool\_variable:", bool\_variable)

print("Тип переменной bool\_variable:", type(bool\_variable)) # Проверка типа переменной

bool\_variable\_2 = True # Используем ключевое слово True для логического значения истины

print("Значение переменной bool\_variable\_2:", bool\_variable\_2)

print("Тип переменной bool\_variable\_2:", type(bool\_variable\_2)) # Проверка типа переменной

**Задание:**

Я работаю в компании, обслуживающей проблемы информационной безопасности на других предприятиях. К нам обратился директор маленькой рекламной компании. Основная проблема в том, что у них есть охранник Дмитрий, который устанавливает компьютерные игры на АРМ (автоматизированные рабочие места) сотрудников, когда тех нет на работе по долгу (находятся в отпуске) и играет всеми ночами напролет. Поэтому вас просят разработать приложение, которое проверяло введенные учетные и разграничивало права сотрудников. Для охранника Дмитрия просят сделать специализированное уведомление: «Дмитрий, твое рабочее место находится в другой комнате. Отойди от чужого компьютера и займись работой!».

1. Введем переменную «user\_name»
2. Далее введем переменную, которая выводит текст для Дмитрия. Назовем ее Dmitriy\_check
3. Введем переменную, которая хранит сообщение для других сотрудников, вошедших в систему: «Добро пожаловать»
4. Напишем оператор if, которые проверяет значение переменной user\_name
5. Вывести в консоль результат выполнения программы для user\_name= «Дмитрий» и для user\_name= «Ангелина».

user\_name = input("Введите ваше имя: ")

Dmitriy\_check = "Дмитрий, твое рабочее место находится в другой комнате. Отойди от чужого компьютера и займись работой!"

welcome\_message = "Добро пожаловать"

if user\_name == "Дмитрий":

print(Dmitriy\_check)

else:

print(welcome\_message)

Наша компания продолжает разрабатывать приложение по безопасности для рекламной компании. Теперь, если пользователь 3 раза ввел пароль неправильно, необходимо заблокировать систему. Для этого выполним следующие действия:

1. Введем переменную, которая фиксирует количество попыток ввода и назовем ее enter\_number
2. С использованием оператора if напишем программу, которая если enter\_number меньше 3, то пишите «Попробуйте еще раз. У вас осталось (3- enter\_number) попыток». Если количество попыток больше либо равно 3, то выводим «Вы превысили максимальное число попыток. Ваша учетная запись заблокирована. Для разблокировки обратитесь в службу поддержки».
3. Проверить работу программы с использованием вывода в консоль.

enter\_number = int(input("Введите количество попыток ввода: "))

if enter\_number < 3:

print(f"Попробуйте еще раз. У вас осталось {3 - enter\_number} попыток.")

else:

print("Вы превысили максимальное число попыток. Ваша учетная запись заблокирована. Для разблокировки обратитесь в службу поддержки.")

**Задание:**

1. Проверить истинность следующих выражений:

(2 + 2 + 2 >= 6) and (-1 \* -1 < 0)

(4 \* 2 <= 8) and (7 - 1 == 6)

Результат проверки поместить в переменные statement\_one и statement\_two.

statement\_one = (2 + 2 + 2 >= 6) and (-1 \* -1 < 0)

statement\_two = (4 \* 2 <= 8) and (7 - 1 == 6)

print("Результат первого выражения:", statement\_one)

print("Результат второго выражения:", statement\_two)

1. Вернемся к задаче с проверкой безопасности в рекламном агентстве. Усложним задачу. Теперь нужно проверять не только имя пользователя, но и номер АРМ.

Для этого введем номера рабочих мест 4-х сотрудников этой компании:

Дмитрий номер АРМ 1

Ангелина номер АРМ 2

Василий номер АРМ 3

Екатерина номер АРМ 4.

1. Введем переменную ARM, обозначающую номер АРМ.
2. Создадим условный оператор if, при котором проверяем соответствие номера АРМ и имени пользователя:

Если номер АРМ и имя пользователя соответствуют, то вывести в консоль «Добро пожаловать!»

Если номер АРМ не совпадает, а имя пользователя не Дмитрий, то «Логин или пароль не верный, попробуйте еще раз»

Если Если номер АРМ не совпадает, а имя пользователя Дмитрий, то Дмитрий, твое рабочее место находится в другой комнате. Отойди от чужого компьютера и займись работой!».

user\_name = input("Введите ваше имя: ")

ARM = int(input("Введите номер АРМ: "))

users = {

"Дмитрий": 1,

"Ангелина": 2,

"Василий": 3,

"Екатерина": 4

}

if user\_name in users:

if ARM == users[user\_name]:

print("Добро пожаловать!")

elif user\_name == "Дмитрий":

print("Дмитрий, твое рабочее место находится в другой комнате. Отойди от чужого компьютера и займись работой!")

else:

print("Логин или пароль не верный, попробуйте еще раз.")

else:

print("Такого пользователя не существует.")

**Задание:**

1. Проверить истинность следующих выражений:

(2 - 1 > 3) or (-5 \* 2 == -10)

(9 + 5 <= 15) or (7 != 4 + 3)

statement\_one = (2 - 1 > 3) or (-5 \* 2 == -10)

statement\_two = (9 + 5 <= 15) or (7 != 4 + 3)

print("Результат первого выражения:", statement\_one)

print("Результат второго выражения:", statement\_two)

**Задания:**

Провести рефакторинг (переделать код) в задаче про Дмитрия и рекламное агентство с использованием оператора else.

user\_name = input("Введите ваше имя: ")

ARM = int(input("Введите номер АРМ: "))

users = {

"Дмитрий": 1,

"Ангелина": 2,

"Василий": 3,

"Екатерина": 4

}

if user\_name in users:

if ARM == users[user\_name]:

print("Добро пожаловать!")

elif user\_name == "Дмитрий":

print("Дмитрий, твое рабочее место находится в другой комнате. Отойди от чужого компьютера и займись работой!")

else:

print("Логин или пароль не верный, попробуйте еще раз.")

else:

print("Такого пользователя не существует.")

Задача про Дмитрия и рекламное агентство:

Нужно проверять не только имя пользователя, но и номер АРМ.

Для этого введем номера рабочих мест 4х сотрудников этой компании:

Дмитрий номер АРМ 1

Ангелина номер АРМ 2

Василий номер АРМ 3

Екатерина номер АРМ 4.

Введем переменную ARM, обозначающую номер АРМ.

Создадим условный оператор if, при котором проверяем соответствие номера АРМ и имени пользователя:

Если номер АРМ и имя пользователя соответствуют, то вывести в консоль «Добро пожаловать!»

Если номер АРМ не совпадает, а имя пользователя не Дмитрий, то «Логин или пароль не верный, попробуйте еще раз»

Если Если номер АРМ не совпадает, а имя пользователя Дмитрий, то Дмитрий, твое рабочее место находится в другой комнате. Отойди от чужого компьютера и займись работой!».

**Код:**

user\_name = input("Введите ваше имя: ")

ARM = int(input("Введите номер АРМ: "))

users = {

"Дмитрий": 1,

"Ангелина": 2,

"Василий": 3,

"Екатерина": 4

}

if user\_name in users:

if ARM == users[user\_name]:

print("Добро пожаловать!")

else:

if user\_name == "Дмитрий":

print("Дмитрий, твое рабочее место находится в другой комнате. Отойди от чужого компьютера и займись работой!")

else:

print("Логин или пароль не верный, попробуйте еще раз.")

else:

print("Такого пользователя не существует.")

**Задание:**

В университете действует система грейдов, которая присваивается студенту в зависимости от среднего балла. Вас просят написать приложение, которое выводило бы соответствующий грейд для каждого студента при следующих условиях:

• 4.0 или выше должен вернуть "A"

• 3.0 или выше должен вернуть "B"

• 2.0 или выше должен вернуть "C"

• 1.0 или выше должен вернуть "D"

• 0,0 или выше должен вернуть "F"

Для решения ввести переменную grade, а также использовать оператор elif.

**Код:**

grade = float(input("Введите средний балл студента: "))

if grade >= 4.0:

print("A")

elif grade >= 3.0:

print("B")

elif grade >= 2.0:

print("C")

elif grade >= 1.0:

print("D")

else:

print("F")  
  
 **Задание**

В университете действует система грейдов, которая присваивается студенту в зависимости от среднего балла. Вас просят написать приложение, которое выводило бы соответствующий грейд для каждого студента при следующих условиях:

• 4.0 или выше должен вернуть "A"

• 3.0 или выше должен вернуть "B"

• 2.0 или выше должен вернуть "C"

• 1.0 или выше должен вернуть "D"

• 0,0 или выше должен вернуть "F"

Для решения ввести переменную grade, а также использовать оператор match\case

grade = float(input("Введите средний балл студента: "))

match grade:

case \_ if grade >= 4.0:

print("A")

case \_ if grade >= 3.0:

print("B")

case \_ if grade >= 2.0:

print("C")

case \_ if grade >= 1.0:

print("D")

case \_:

print("F")  
  
  
 **Задание**

1. Определите функцию с именем create\_spreadsheet (), которая принимает один аргумент title и выводит в консоль только одну строку «Создание электронной таблицы с именем» + title.

2. Вызовите create\_spreadsheet с значением title «Загрузки».

3. Добавьте параметр row\_count в определение функции. Установите значение по умолчанию 1000.

4. Измените оператор print «Создание электронной таблицы с названием title with row\_count lines», где title и row\_count заменяются их соответствующими значениями.

Помните, чтобы объединить число в строковый объект, вам сначала нужно привести row\_count к строке, используя str (). В противном случае вы получите ошибку TypeError.

5. Вызовите create\_spreadsheet () с названием набора в «Приложения» и row\_countset в 10.

**Код:**

def create\_spreadsheet(title):

print("Создание электронной таблицы с именем " + title)

create\_spreadsheet("Загрузки")

def create\_spreadsheet(title, row\_count=1000):

print("Создание электронной таблицы с названием " + title + " with " + str(row\_count) + " lines")

create\_spreadsheet("Приложения", 10)

**Задание**

1. Функция define\_age создает переменную с именем age, которая представляет собой разницу между текущим годом и годом рождения, оба из которых являются входными данными для функции. Добавьте строку, чтобы возраст возвращался в качестве результата вычислений для дальнейшего использования.

>>def calc\_age (current\_year, birth\_year):

#возраст = текущий\_год - рождение\_год

2. Вне функции вызовите Calculate\_age со значениями 2049 (current\_year) и 1993 (birth\_year) и сохраните значение в переменную my\_age.

3. Вызовите Calculate\_age со значениями 2049 (current\_year) и 1953 (birth\_year) и сохраните значение в переменной с именем dads\_age.

Выведите на консоль строку «Мне X лет, а моему отцу Y лет», с my\_age, где X и dads\_age, где Y.

**Код:**

def calc\_age(current\_year, birth\_year):

age = current\_year - birth\_year

return age

my\_age = calc\_age(2049, 1993)

dads\_age = calc\_age(2049, 1953)

print(f"Мне {my\_age} лет, а моему отцу {dads\_age} лет.")

**Задание**

1. Напишите функцию с именем get\_boundaries (), которая принимает два параметра: числовой параметр target и числовой параметр margin.

Следует создать две переменные:

• low\_limit: target минус margin.

• high\_limit: margin прибавить к target

2.Возвратить значения low\_limit и high\_limit из функции в указанном порядке.

3.Вызовите функцию с параметром target, равным 100 и с margin 20. Сохраните возвращаемые значения в переменные, называемые low\_limit и high\_limit.

4.Выведите в консоль строку:

Нижний предел: low\_limit, верхний предел: high\_limit

Со значениями low и high, которые вы получили из функции get\_boundaries ().

def get\_boundaries(target, margin):

low\_limit = target - margin

high\_limit = target + margin

return low\_limit, high\_limit

low\_limit, high\_limit = get\_boundaries(100, 20)

print(f"Нижний предел: {low\_limit}, верхний предел: {high\_limit}")

**Задание**

1. Определите функцию с именем repeat\_stuff, которая принимает два входа, stuff и num\_repeats.

Мы хотим, чтобы эта функция вывела в консоль строку с количеством повторений nre\_repeats. Пока только поместите пустой оператор print внутри функции.

1. Вне функции вызовите repeat\_stuff.

Вы можете использовать значение "Row" для stuff и 3 для num\_repeats.

1. Измените оператор вывода внутри repeat\_stuff на returnstatement.

Он должен возвращать stuff \* num\_repeats.

Примечание: умножение строки просто приводит к повторению значения строки! Например:

"na"\*6

приводит к строке "nananananana".

1. Присвойте параметру num\_repeats значение по умолчанию 10.

5. Соедините результат repeat\_stuff ("Row", 3) и строку "Your Boat." вместе и сохраните результат в переменную с именем lyrics.

6. Создайте переменную с именем song и присвойте ей значение repeat\_stuff, вызываемого с только с stuff.

7. Вывести песню в консоль.

def repeat\_stuff(stuff, num\_repeats=10):

return stuff \* num\_repeats

lyrics = repeat\_stuff("Row", 3) + " Your Boat."

song = repeat\_stuff("Row") # Используется значение по умолчанию для num\_repeats

print(song)

**Задание**

1. Создайте список, который содержит наименование товара «торт» и цифру 1.

product\_list = ["торт", 1]

print(product\_list)

**Задание**

1. Нам написали дополнительный список товаров, которые необходимо купить в отделе бытовой химии. Он включает в себя стиральный порошок в количестве 1 штуки и средство для мытья посуды так же 1 штуку. Создайте новый список household\_chemicals, который будет содержать список, состоящий из списков, в которых первым элементом будет наименование товара, а вторым количество.

household\_chemicals = [

["стиральный порошок", 1],

["средство для мытья посуды", 1]

]

print(household\_chemicals)

**Задание**

1. Используйте zip для создания новой переменной с именем names\_and\_dogs\_names, которая объединяет имен а и имена собак в zip-объект.

Names=[‘Ben’, ‘Holly’, ‘Ann’]

dogs\_names= [‘Sharik’, ‘Gab’, ‘Beethoven’]

1. Создайте новую переменную с именем list\_of\_names\_and\_dogs\_names, вызвав list () для names\_and\_dogs\_names. Выведите в консоль новую переменную.

names = ['Ben', 'Holly', 'Ann']

dogs\_names = ['Sharik', 'Gab', 'Beethoven']

names\_and\_dogs\_names = zip(names, dogs\_names)

list\_of\_names\_and\_dogs\_names = list(names\_and\_dogs\_names)

print(list\_of\_names\_and\_dogs\_names)

**Задание**

1. Мария работает в цветочном магазине. Она ведет учет заказов в списке, который называется orders.

Создайте список orders и используйте print, чтобы проверить заказы, которые он получил сегодня.

orders = ['маргаритки', 'васильки'].

2. Мария только что получила новый заказ на тюльпаны. Используйте append, чтобы добавить эту строку в orders.

3. Пришел еще один заказ! Используйте append, чтобы добавлять «розы» к заказам.

4. Используйте print, чтобы просмотреть заказы, которые получила сегодня Мария.

orders = ['маргаритки', 'васильки']

print("Заказы, полученные сегодня:", orders)

orders.append('тюльпаны')

orders.append('розы')

print("Обновленные заказы, полученные сегодня:", orders)

**Задание**

1. Мария все еще обновляет свой список заказов. Она только что получила заказы на «сирень» и «ирис».

Используйте +, чтобы создать новый список с именем new\_orders, который объединяет заказы с двумя новыми заказами.

orders = ['маргаритка', 'лютик', 'львиный зев', 'гардения', 'лилия']

# broken\_prices = [5, 3, 4, 5, 4] + 4

1. Удалите # перед списком broken\_prices. Если вы запустите этот код, вы получите сообщение об ошибке
2. Исправьте команду, чтобы она выполнялась без ошибок.

orders = ['маргаритка', 'лютик', 'львиный зев', 'гардения', 'лилия']

new\_orders = orders + ['сирень', 'ирис'] # Объединяем списки

print("Обновленные заказы:", new\_orders)

broken\_prices = [5, 3, 4, 5, 4] # Убираем комментарий

print("Цены на товары:", broken\_prices)

**Задание**

1. Измените list1 так, чтобы это был диапазон, содержащий числа, начинающиеся с 0 и до 9, но не включая 9.

list1=[1,8]

1. Создайте диапазон с именем list2 с числами от 0 до 7.

list1 = list(range(0, 9)) # Создаем список от 0 до 8

list2 = list(range(0, 8)) # Создаем список от 0 до 7

print("list1:", list1)

print("list2:", list2)

**Задание**

1. Измените функцию диапазона, создавшую list1, так, чтобы она:

• Начиналась с 5

• Разница между каждым элементом составляет 3 единицы.

• Заканчивается на 15

list1 = диапазон (6, 15, 2)

1. Создайте объект диапазона с именем list2, который:

• Начинается с 0

• Разница между каждым элементом составляет 5 единиц.

• Заканчивается до 40

list1 = list(range(5, 16, 3)) # 16 не включается

list2 = list(range(0, 40, 5)) # 40 не включается

print("list1:", list1)

print("list2:", list2)

**Задания**

1. Мария вводит данные клиентов для своего бизнеса в области веб-дизайна. Вы поможете ей организовать свои данные.

Начните с превращения этого списка имен клиентов в список с именем first\_names. Обязательно вводите имена в таком порядке:

• Анна

• Борис

• Александр

• Денис

2. Создайте пустой список под названием age.

3. Возраст Дениса - 42 года. Используйте .append (), чтобы прибавить 42 к age.

4. Марии нужен список возрастов для всех клиентов. Создайте новый список с именем all\_ages, который добавляет возраст со следующим списком, содержащим возраст Анны, Бориса и Александра:

[32, 41, 29]

Убедитесь, что all\_ages начинается с возраста Анны, Бориса и Александра и заканчивается возрастом Дениса (хранится в age)!

5. Создайте новую переменную name\_and\_age, которая объединяет first\_names и all\_ages с помощью zip.

6. Создайте диапазон с именем ids с идентификационным номером для каждого клиента. Поскольку клиентов 4, значения id должны изменяться от 0 до 3.

first\_names = ['Анна', 'Борис', 'Александр', 'Денис']

age = []

age.append(42)

all\_ages = [32, 41, 29] # Возраст Анны, Бориса и Александра

all\_ages.append(age[0]) # Добавляем возраст Дениса

name\_and\_age = list(zip(first\_names, all\_ages))

ids = list(range(len(first\_names))) # Идентификационные номера от 0 до 3

print("Имена и возрасты:", name\_and\_age)

print("Идентификационные номера:", ids)

---------------------------------------------------------------------------------------------------------------------

**Задание**

1. Вычислите длину list1 и сохраните ее в переменной list1\_len.

list1 = range (2, 20, 2)

1. Используйте print для проверки list1\_len.

# Генерируем список с шагом 2

list1 = range(2, 20, 2)

# Вычисляем длину списка

list1\_len = len(list1)

# Выводим длину списка

print("Длина list1:", list1\_len)

3. Измените команду range, которая генерирует list1, чтобы она пропускала 3 вместо 2 элемента.

Как это изменит list1\_len?

# Генерируем список с шагом 3

list1 = range(2, 20, 3)

# Вычисляем длину нового списка

list1\_len = len(list1)

# Выводим длину нового списка

print("Длина list1 при шаге 3:", list1\_len)

**Задание**

1. Используйте print и len, чтобы отобразить длину shopping\_list.

shopping\_list = ['яйца', 'масло', 'молоко', 'огурцы', 'сок', 'хлопья']

2. Получите последний элемент shopping\_list, используя индекс -1. Сохраните этот элемент в переменной last\_element.

3. Теперь выберите элемент с индексом 5 и сохраните его в переменной element5.

4. Используйте print для отображения как element5, так и last\_element.

# Список покупок

shopping\_list = ['яйца', 'масло', 'молоко', 'огурцы', 'сок', 'хлопья']

# Отображаем длину списка

print("Длина списка покупок:", len(shopping\_list))

# Получаем последний элемент списка

last\_element = shopping\_list[-1]

# Отображаем последний элемент

print("Последний элемент списка:", last\_element)  
  
  
# Получаем элемент с индексом 5

element5 = shopping\_list[5]

# Отображаем элемент с индексом 5

print("Элемент с индексом 5:", element5)  
  
  
# Отображаем оба элемента

print("Элемент с индексом 5:", element5)

print("Последний элемент:", last\_element)

**Задание**

1. Используйте print, чтобы проверить переменную beginning.

suitcase = ['рубашка', 'рубашка', 'брюки', 'брюки', 'пижамы', 'книги']

beginning = suitcase [0: 2]

Сколько элементов в списке?

2. Измените beginning, чтобы оно выделяло первые 4 элемента чемодана.

3. Создайте новый список под названием middle, содержащий два средних элемента из чемодана.

# Список вещей в чемодане

suitcase = ['рубашка', 'рубашка', 'брюки', 'брюки', 'пижамы', 'книги']

# Первые 2 элемента списка

beginning = suitcase[0:2]

# Отображаем переменную beginning

print("Первые два элемента:", beginning)

# Количество элементов в beginning

print("Количество элементов в beginning:", len(beginning))  
  
  
  
# Первые 4 элемента списка

beginning = suitcase[0:4]

# Отображаем переменную beginning

print("Первые четыре элемента:", beginning)

# Список со средними элементами (индексы 2 и 3)

middle = suitcase[2:4]

# Отображаем переменную middle

print("Средние два элемента:", middle)

**Задание**

Создайте новый список под названием start, содержащий первые 3 элемента чемодана.

suitcase = ['рубашка', 'футболка', 'носки', 'очки', 'пижама', 'книги']

# Список вещей в чемодане

suitcase = ['рубашка', 'футболка', 'носки', 'очки', 'пижама', 'книги']

# Создаем новый список start, содержащий первые 3 элемента

start = suitcase[:3]

# Отображаем переменную start

print("Первые три элемента:", start)

**Задание**

Класс миссис Уилсон голосует за президента класса. Она сохранила голос каждого студента в списке голосов.

Используйте счетчик, чтобы определить, сколько студентов проголосовало за «Jake». Сохраните свой ответ как jake\_votes.

votes = ['Jake', 'Jake', 'Laurie', 'Laurie', 'Laurie', 'Jake', 'Jake', 'Jake', 'Laurie', 'Cassie', 'Cassie', 'Jake', 'Jake', 'Cassie', 'Laurie', 'Cassie', 'Jake', 'Jake', 'Cassie', 'Laurie']

Используйте print для проверки jake\_votes.

# Список голосов

votes = ['Jake', 'Jake', 'Laurie', 'Laurie', 'Laurie', 'Jake', 'Jake', 'Jake',

'Laurie', 'Cassie', 'Cassie', 'Jake', 'Jake', 'Cassie', 'Laurie',

'Cassie', 'Jake', 'Jake', 'Cassie', 'Laurie']

# Используем метод count для подсчета голосов за 'Jake'

jake\_votes = votes.count('Jake')

# Отображаем количество голосов за 'Jake'

print("Количество голосов за Jake:", jake\_votes)

**Задание**

Отсортируйте адреса.

addresses = ['221 B Baker St.', '42 Wallaby Way', '12 Grimmauld Place', '742 Evergreen Terrace', '1600 Pennsylvania Ave', '10 Downing St.']

Используйте print, чтобы увидеть, как менялись addresses

# Список адресов

addresses = ['221 B Baker St.', '42 Wallaby Way', '12 Grimmauld Place',

'742 Evergreen Terrace', '1600 Pennsylvania Ave', '10 Downing St.']

# Сортировка адресов

addresses.sort()

# Вывод отсортированного списка

print("Отсортированные адреса:")

print(addresses)

**Задание**

1. Используйте сортировку, чтобы упорядочить игры и создать новый список с именем games\_sorted.

games = ['Portal', 'Minecraft', 'Pacman', 'Tetris', 'The Sims', 'Pokemon']

Используйте print для проверки игр и games\_sorted.   
  
  
# Список игр

games = ['Portal', 'Minecraft', 'Pacman', 'Tetris', 'The Sims', 'Pokemon']

# Создание нового списка с отсортированными играми

games\_sorted = sorted(games)

# Вывод исходного и отсортированного списков

print("Исходный список игр:")

print(games)

print("\nОтсортированный список игр:")

print(games\_sorted)

**Задание**

1. inventory - это список предметов, которые есть на складе мебели. Сколько товаров на складе?

Сохраните свой ответ в inventory\_len.

inventory = [«двухспальная кровать», «двухспальная кровать», «изголовье», «двуспальная кровать», «двуспальная кровать», «комод», «комод», «стол», «стол», «тумбочка», «тумбочка», «королевский кровать», «двуспальная кровать», «две односпальные кровати», «две односпальные кровати», «простыни», «простыни», «подушка», «подушка»]

2. Выберите первый элемент в инвентаре. Сначала сохраните его в переменной first.

3. Выберите последний элемент из инвентаря и сохраните его в переменной last.

4. Выберите предметы из инвентаря, начиная с индекса 2 и до индекса 6, но не включая его.

Сохраните свой ответ в inventory\_2\_6.

1. Выберите первые 3 предмета инвентаря и сохраните их в переменной first\_3.
2. Сколько односпальных кроватей в инвентаре? Сохраните свой ответ в twin\_beds.
3. Сортировка инвентаря с помощью .sort ().

# Инвентарь

inventory = [

"двуспальная кровать", "двуспальная кровать", "изголовье", "двуспальная кровать",

"двуспальная кровать", "комод", "комод", "стол", "стол", "тумбочка",

"тумбочка", "королевская кровать", "двуспальная кровать",

"две односпальные кровати", "две односпальные кровати", "простыни",

"простыни", "подушка", "подушка"

]

# 1. Сколько товаров на складе?

inventory\_len = len(inventory)

print("Количество товаров на складе:", inventory\_len)

# 2. Выберите первый элемент в инвентаре

first = inventory[0]

# 3. Выберите последний элемент из инвентаря

last = inventory[-1]

# 4. Выберите предметы из инвентаря с индекса 2 до 6 (не включая 6)

inventory\_2\_6 = inventory[2:6]

# 5. Выберите первые 3 предмета инвентаря

first\_3 = inventory[:3]

# 6. Сколько односпальных кроватей в инвентаре?

twin\_beds = inventory.count("односпальная кровать") # Обратите внимание, что "односпальная кровать" не была в списке, поэтому нужно использовать "две односпальные кровати"

twin\_beds = inventory.count("две односпальные кровати") \* 2 # Учитываем, что в инвентаре два раза указана односпальная кровать

# 7. Сортировка инвентаря

inventory.sort()

# Проверка результатов

print("\nПервый элемент:", first)

print("Последний элемент:", last)

print("Элементы с индекса 2 до 6:", inventory\_2\_6)

print("Первые 3 предмета инвентаря:", first\_3)

print("Количество односпальных кроватей:", twin\_beds)

print("Отсортированный инвентарь:", inventory)

**Задание**

1. Марк работает официантом в ресторане. Он принял заказ и внес его в список. Создайте список order и внесите в него пасту, пиццу и салат капрезе.
2. К столику присоединился еще один посетитель и Марк принял у него заказ, состоящий из салата цезарь и кофе. Добавьте в конец списка order данные позиции

Клиенты вспомнили, что совсем забыли про напитки! Добавьте в список заказа красное сухое вино.  
  
  
# 1. Создание списка заказов

order = ['паста', 'пицца', 'салат капрезе']

# 2. Добавление новых позиций в заказ

order.append('салат цезарь')

order.append('кофе')

# 3. Добавление красного сухого вина в заказ

order.append('красное сухое вино')

# Проверка окончательного списка заказов

print("Список заказов:", order)

**Задание**

Список с заказом был передан Марком на кухню, однако клиент попросил принести закуску из овощей перед тем, как принесут весь заказ. Добавьте в orders данную позицию в начало списка.  
  
  
# Изначальный список заказов

orders = ['паста', 'пицца', 'салат капрезе', 'салат цезарь', 'кофе', 'красное сухое вино']

# 1. Добавление закуски из овощей в начало списка

orders.insert(0, 'закуска из овощей')

# Проверка окончательного списка заказов

print("Обновленный список заказов:", orders)

**Задание**

1. Клиенты решили, что слишком много еды и решили отказаться от салата капрезе. Удалите данную позицию из списка

# Изначальный список заказов

orders = ['паста', 'пицца', 'салат капрезе', 'салат цезарь', 'кофе', 'красное сухое вино']

# 1. Удаление салата капрезе из списка

orders.remove('салат капрезе')

# Проверка окончательного списка заказов

print("Обновленный список заказов:", orders)

**Задание**

1. Клиенты Марка вспомнили, что сегодня у них запланирован важный созвон поэтому передумали пить вино. Они попросили Марка удалить этот пункт из заказа. Удалите вино из исходного списка с заказом.

# Изначальный список заказов

orders = ['паста', 'пицца', 'салат капрезе', 'салат цезарь', 'кофе', 'красное сухое вино']

# 1. Удаление красного сухого вина из списка

orders.remove('красное сухое вино')

# Проверка окончательного списка заказов

print("Обновленный список заказов:", orders)

**Задание**

1. Создайте список чисел от 0 до 7
2. Удалите 2 элемента из середины списка, используя del

# 1. Создание списка чисел от 0 до 7

numbers = list(range(8))

# 2. Удаление двух элементов из середины списка

# Определяем индекс середины

mid\_index = len(numbers) // 2

# Удаляем элементы с индексами mid\_index и mid\_index + 1

del numbers[mid\_index:mid\_index + 2]

# Проверка окончательного списка

print("Обновленный список чисел:", numbers)

**Задание**

1. Выведите список в обратном порядке

x = [15, 11, 13, 12, 14, 10]

# Исходный список

x = [15, 11, 13, 12, 14, 10]

# Вывод списка в обратном порядке

x\_reversed = x[::-1]

print("Список в обратном порядке:", x\_reversed)

**Задание**

1.Запустите код ниже. Вы должны получить ошибку IndentationError, потому что строка печати (игра) не имеет отступа.

board\_games = ['Settlers of Catan', 'Carcassone', 'Power Grid', 'Agricola', 'Scrabble']

sport\_games = ['football', 'football - American', 'hockey', 'baseball', 'cricket']

for game in board\_games:

print(game)

2.Сделайте отступ в строке 6, чтобы при запуске кода не возникала ошибка IndentationError.

3. Напишите цикл, который выводит каждый вид спорта в sport\_games.  
  
  
# Списки настольных и спортивных игр

board\_games = ['Settlers of Catan', 'Carcassonne', 'Power Grid', 'Agricola', 'Scrabble']

sport\_games = ['football', 'football - American', 'hockey', 'baseball', 'cricket']

# Цикл для вывода настольных игр

for game in board\_games:

print(game) # Исправлен отступ здесь

# Цикл для вывода спортивных игр

for sport in sport\_games:

print(sport)

**Задание**

1. Используйте функцию диапазона в цикле for, чтобы распечатать обещание 5 раз.

promise = "I will not chew gum in class"

promise = "I will not chew gum in class"

# Используем цикл for с диапазоном от 0 до 4 (всего 5 раз)

for \_ in range(5):

print(promise)

**Задание**

1. Предположим, у нас есть два списка студентов: student\_period\_A и student\_period\_B. Мы хотим объединить всех студентов в student\_period\_B.

Напишите цикл for, который проходит через каждого студента в student\_period\_A и добавляет его в конец student\_period\_B.

students\_period\_A = ["Alex", "Briana", "Cheri", "Daniele"]

students\_period\_B = ["Dora", "Minerva", "Alexa", "Obie"]

2. Внутри цикла for, после добавления student к student\_period\_B, выведите student.

3. Предположим, вы допустили опечатку в теле цикла for.

Внутри цикла for измените объект оператора добавления с student\_period\_B на student\_period\_A. В данном случае вами будет получен бесконечный цикл

Выйдите из бесконечного цикла! Затем избавьтесь от ошибки, которая вызвала бесконечный цикл.

# Списки студентов

students\_period\_A = ["Alex", "Briana", "Cheri", "Daniele"]

students\_period\_B = ["Dora", "Minerva", "Alexa", "Obie"]

# Объединение студентов из period A в period B

for student in students\_period\_A:

students\_period\_B.append(student) # Добавляем студента в список period B

print(student) # Выводим имя добавленного студента

# Проверим результат объединения

print("Обновленный список студентов в периоде B:")

print(students\_period\_B)

# Ошибка с бесконечным циклом (для демонстрации)

# Поскольку это приведет к бесконечному циклу, я использую переменную count для выхода из него

count = 0

for student in students\_period\_A:

students\_period\_A.append(student) # Ошибка: добавляем студента в тот же список

count += 1

if count > 10: # Ограничиваем количество добавлений для выхода из бесконечного цикла

break

# После выхода из цикла исправим ошибку

# Верное использование: добавление студента в students\_period\_B

for student in students\_period\_A:

students\_period\_B.append(student) # Правильное добавление

print(student) # Выводим имя добавленного студента

**Задание**

1. У вас есть список пород собак, из числа которых вы можете взять себе питомца, dog\_breeds\_available\_for\_adoption. Используя цикл for, выполните итерацию по списку dog\_breeds\_available\_for\_adoption и распечатайте каждую породу собак.

dog\_breeds\_available\_for\_adoption = ['french\_bulldog', 'dalmatian', 'shihtzu', 'poodle', 'collie']

dog\_breed\_I\_want = 'dalmatian'

1. Внутри цикла for проверьте, совпадает ли текущее значение породы с dog\_breed\_I\_want. Если да, то выведите "У них есть собака, которую я хочу!"

Добавьте оператор break, когда ваш цикл обнаружит dog\_breed\_I\_want, чтобы не нужно было проверять остальную часть списка.  
  
  
# Список доступных пород собак для усыновления

dog\_breeds\_available\_for\_adoption = ['french\_bulldog', 'dalmatian', 'shihtzu', 'poodle', 'collie']

dog\_breed\_I\_want = 'dalmatian'

# Итерация по списку пород собак

for breed in dog\_breeds\_available\_for\_adoption:

print(breed) # Выводим каждую породу собаки

# Проверка, совпадает ли текущая порода с желаемой

if breed == dog\_breed\_I\_want:

print("У них есть собака, которую я хочу!") # Уведомляем, если порода совпадает

break # Выходим из цикла, чтобы не продолжать проверки

**Задание**

Мы предоставили список sales\_data, который показывает количество различных вкусов мороженого, проданных в трех разных местах вымышленного магазина. Мы хотим суммировать общее количество проданных сортов. Начните с определения переменной scoops\_sold и установите ее равной нулю.

sales\_data = [[12, 17, 22], [2, 10, 3], [5, 12, 13]]

2. Просмотрите список sales\_data. Создайте цикл, перебирающий каждый список из списка sales\_data .

3.В списке sales\_data переберите значения внутри каждого вложенного списка из sales\_data  и прибавьте к своей переменной scoops\_sold.

К концу у вас должна быть сумма всех чисел во вложенном списке sales\_data.

1. Выведите значение scoops\_sold.

# Список доступных пород собак для усыновления

dog\_breeds\_available\_for\_adoption = ['french\_bulldog', 'dalmatian', 'shihtzu', 'poodle', 'collie']

dog\_breed\_I\_want = 'dalmatian'

# Итерация по списку пород собак

for breed in dog\_breeds\_available\_for\_adoption:

print(breed) # Выводим каждую породу собаки

# Проверка, совпадает ли текущая порода с желаемой

if breed == dog\_breed\_I\_want:

print("У них есть собака, которую я хочу!") # Уведомляем, если порода совпадает

break # Выходим из цикла, чтобы не продолжать проверки

**Задание**

1. Создайте список с именем single\_digits, состоящий из чисел от 0 до 9 (включительно).

2. Создайте цикл for, который проходит через single\_digits и выводит каждую из них.

3. Перед циклом создайте список под названием squares. Назначьте для начала пустой список.

4. Внутри цикла, который повторяет single\_digits, добавьте значение квадрата каждого элемента single\_digits к squares. Вы можете сделать это до или после вывода элемента.

5. После цикла for выведите squares.

6. Создайте список cubes, используя сравнение списков в списке single\_digits. Каждый элемент cubes должен быть элементом single\_digits, возведенным в куб.

7. Выведите cubes.  
  
  
# 1. Создание списка single\_digits

single\_digits = list(range(10)) # Числа от 0 до 9

# 2. Цикл for, который проходит через single\_digits и выводит каждую из них

for digit in single\_digits:

print(digit) # Выводим каждую цифру

# 3. Создание списка squares и инициализация его пустым списком

squares = []

# 4. Внутри цикла добавление квадрата каждого элемента single\_digits к squares

for digit in single\_digits:

squares.append(digit \*\* 2) # Добавляем квадрат текущего элемента

# 5. Вывод squares после цикла

print("Квадраты:", squares)

# 6. Создание списка cubes, где каждый элемент — это куб соответствующего элемента single\_digits

cubes = [digit \*\* 3 for digit in single\_digits]

# 7. Вывод cubes

print("Кубы:", cubes)

**Задание**

1. Сохраните ваше любимое слово в виде строки в переменной favour\_word.

2. Выведите favour\_word.

# 1. Сохраните ваше любимое слово в переменной

favour\_word = "ваше\_любимое\_слово" # Замените на ваше слово

# 2. Выведите favour\_word

print(favour\_word)

**Задание**

1. Вы программист, работаете в большой компании. В этой компании имя пользователя каждого сотрудника создается путем взятия первых пяти букв их фамилии.

Новый сотрудник, Виталий Красилов, начинает работу сегодня, и вам необходимо создать его учетную запись. Запишите его first\_name и last\_name.

Создайте переменную new\_account, срезав первые пять букв его last\_name.

1. Временные пароли для новых сотрудников также генерируются из их фамилий.

Создайте переменную с именем temp\_password, создав фрагмент с третьей по шестую буквы его last\_name.

# 1. Запишите имя и фамилию нового сотрудника

first\_name = "Виталий"

last\_name = "Красилов"

# Создайте учетную запись

new\_account = last\_name[:5] # Первые пять букв фамилии

# 2. Создайте временный пароль

temp\_password = last\_name[2:6] # С третьей по шестую буквы фамилии

# Вывод результатов

print("Учетная запись:", new\_account)

print("Временный пароль:", temp\_password)

**Задание**

1. Руководство Компании, в которой вы работаете, осознало, что их политика использования первых пяти букв фамилии сотрудника в качестве имени пользователя не идеальна, если у них несколько сотрудников с одинаковой фамилией.

Напишите функцию с именем account\_generator, которая принимает два входа, first\_name и last\_name, объединяет первые три буквы каждого из них, а затем возвращает новое имя учетной записи.

Проверьте свою функцию и сохраните результат ее выполнения в переменную new\_account.  
  
  
# 1. Определите функцию для генерации имени учетной записи

def account\_generator(first\_name, last\_name):

# Объединяем первые три буквы имени и фамилии

return first\_name[:3] + last\_name[:3]

# 2. Проверьте функцию и сохраните результат

first\_name = "Виталий"

last\_name = "Красилов"

new\_account = account\_generator(first\_name, last\_name)

# Вывод результата

print("Новая учетная запись:", new\_account)

**Задание**

1. Руководство компании снова хочет обновить способ генерации временных паролей для новых сотрудников.

Напишите функцию с именем password\_generator, которая принимает два входа, first\_name и last\_name, а затем объединяет последние три буквы каждого из них и возвращает их в виде строки.

1. Протестируйте свою функцию и сохраните их в переменной temp\_password.

# 1. Определите функцию для генерации временного пароля

def password\_generator(first\_name, last\_name):

# Объединяем последние три буквы имени и фамилии

return first\_name[-3:] + last\_name[-3:]

# 2. Проверьте функцию и сохраните результат

first\_name = "Виталий"

last\_name = "Красилов"

temp\_password = password\_generator(first\_name, last\_name)

# Вывод результата

print("Временный пароль:", temp\_password)

**Задание**

1. Используйте отрицательные индексы, чтобы найти предпоследний символ в company\_motto. Сохраните это в переменной second\_to\_last.

company\_motto= «Мечты сбываются»

1. Используйте отрицательные индексы, чтобы создать фрагмент из последних 4 символов в company\_motto. Сохраните это в переменной final\_word.

# 1. Определите строку company\_motto

company\_motto = "Мечты сбываются"

# Найдите предпоследний символ

second\_to\_last = company\_motto[-2]

# 2. Создайте фрагмент из последних 4 символов

final\_word = company\_motto[-4:]

# Вывод результатов

print("Предпоследний символ:", second\_to\_last)

print("Последние 4 символа:", final\_word)

**Задание**

1. Последним сотрудником нашей компании стал парень по имени Роб Дейли. К сожалению, отдел кадров, похоже, допустил небольшую опечатку и прислал неправильное имя first\_name.

Попробуйте изменить первый символ first\_name, запустив

first\_name[0] = "Р"

1. Строки неизменяемы, поэтому мы не можем изменить отдельный символ. Хорошо, это не проблема, мы все еще можем это исправить!

Объедините строку «Р» с фрагментом first\_name, который включает все, кроме первого символа, и сохраните его в новой строке fixed\_first\_name.

# 1. Определите переменную first\_name с ошибкой

first\_name = "Rob"

# Попытка изменить первый символ (это вызовет ошибку)

# first\_name[0] = "Р" # Это вызовет ошибку

# 2. Создайте новую строку с исправленным первым символом

fixed\_first\_name = "Р" + first\_name[1:]

# Вывод результата

print("Исправленное имя:", fixed\_first\_name)

**Задание**

1. Когда Роб Дейли настраивал свою учетную запись, он установил свой пароль:

theycallme"crazy"91

Его пароль вызывал некоторые ошибки в системе из-за отметок ". Перепишите его пароль, используя escape-символы, и сохраните его с переменным паролем.

# Переписываем пароль с использованием escape-символов

password = "theycallme\"crazy\"91"

# Вывод результата

print("Пароль Роб Дейли:", password)

**Задание**

1. Давайте воспроизведем уже знакомую вам функцию len ().

Напишите новую функцию с именем get\_length (), которая принимает строку в качестве входных данных и возвращает количество символов в этой строке. Сделаем это, перебирая строку, используя len ()!

def get\_length(input\_string):

# Перебираем строку и используем len() для получения количества символов

count = 0

for \_ in input\_string: # Проходим по каждому символу в строке

count += 1 # Увеличиваем счетчик на 1 для каждого символа

return count # Возвращаем общее количество символов

# Пример использования функции

my\_string = "Привет, мир!"

length = get\_length(my\_string)

print("Длина строки:", length)

**Задание**

1. Напишите функцию с именем letter\_check, которая принимает два ввода: слово и букву.

Эта функция должна возвращать True, если слово содержит букву, и False, если нет.

def letter\_check(word, letter):

# Проверяем, содержится ли буква в слове

return letter in word

# Пример использования функции

my\_word = "Программирование"

my\_letter = "г"

result = letter\_check(my\_word, my\_letter)

print(f"Слово '{my\_word}' содержит букву '{my\_letter}':", result)

my\_letter2 = "а"

result2 = letter\_check(my\_word, my\_letter2)

print(f"Слово '{my\_word}' содержит букву '{my\_letter2}':", result2)

**Задание**

1. Напишите функцию с именем contains, которая принимает два аргумента, big\_string и little\_string, и возвращает True, если big\_string содержит little\_string.

Например, contains ("watermelon", "melon») должен возвращать True, а contains ("watermelon", "berry») должен возвращать False.

1. Напишите функцию с именем common\_letters, которая принимает два аргумента, string\_one и string\_two, а затем возвращает список со всеми их общими буквами.

Буквы в возвращаемом списке должны быть уникальными. Например,

common\_letters("banana", "cream")

должен вернуть ['a'].

def contains(big\_string, little\_string):

# Проверяем, содержится ли little\_string в big\_string

return little\_string in big\_string

# Примеры использования функции contains

print(contains("watermelon", "melon")) # Должно вернуть True

print(contains("watermelon", "berry")) # Должно вернуть False

def common\_letters(string\_one, string\_two):

# Находим общие буквы, преобразовав строки в множества

common = set(string\_one) & set(string\_two)

# Возвращаем список уникальных букв

return list(common)

# Пример использования функции common\_letters

result = common\_letters("banana", "cream")

print(result) # Должно вернуть ['a']

**Задание**

1. Руководство компании снова заручилось вашей помощью, чтобы создать функцию для создания имени пользователя и пароля. В этом упражнении вы создадите две функции: username\_generator и password\_generator.

Начнем с username\_generator. Создайте функцию с именем username\_generator, которая принимает два входа, first\_name и last\_name, и возвращает имя пользователя. Имя пользователя должно состоять из первых трех букв их имени и первых четырех букв их фамилии. Если их имя состоит менее чем из трех букв или их фамилия менее четырех букв, следует использовать их имена полностью.

Например, если сотрудника зовут Abe Simpson, функция должна сгенерировать имя пользователя AbeSimp.

1. Теперь что касается временного пароля, они хотят, чтобы функция взяла введенное имя пользователя и сдвинула все буквы на одну вправо, чтобы последняя буква имени пользователя стала первой буквой и так далее. Например, если имя пользователя - AbeSimp, то сгенерированный временный пароль должен быть pAbeSim.

Начните с определения функции password\_generator так, чтобы она принимала один ввод, имя пользователя и определяла в нем пустую строку с именем password.

1. Внутри password\_generator создайте цикл for, который перебирает символы в имени пользователя.

def username\_generator(first\_name, last\_name):

# Проверяем длину имени и фамилии

if len(first\_name) < 3:

username\_first\_part = first\_name

else:

username\_first\_part = first\_name[:3]

if len(last\_name) < 4:

username\_second\_part = last\_name

else:

username\_second\_part = last\_name[:4]

# Создаем имя пользователя

return username\_first\_part + username\_second\_part

# Примеры использования функции username\_generator

print(username\_generator("Abe", "Simpson")) # Должно вернуть AbeSimp

print(username\_generator("Al", "Li")) # Должно вернуть AlLi

def password\_generator(username):

password = ""

# Сдвигаем все буквы на одну вправо

if username: # Проверяем, что username не пустой

# Добавляем последнюю букву в начало пароля

password += username[-1]

# Добавляем все буквы кроме последней

for char in username[:-1]:

password += char

return password

# Пример использования функции password\_generator

generated\_username = username\_generator("Abe", "Simpson")

print(password\_generator(generated\_username)) # Должно вернуть pAbeSim

**Задание**

1. Вы программист, работающий в организации, которая пытается оцифровать и хранить стихи под названием «Preserve the Verse».

Вам дали две строки, название стихотворения и автор, и попросили их немного переформатировать, чтобы они соответствовали правилам базы данных организации.

Сделайте poem\_title заголовком и сохраните его в poem\_title\_fixed.

poem\_title = "spring storm"

poem\_author = "William Carlos Williams"

1. Выведите poem\_title и poem\_title\_fixed.

# Исходные строки

poem\_title = "spring storm"

poem\_author = "William Carlos Williams"

# Форматирование заголовка стихотворения

poem\_title\_fixed = poem\_title.title() # Приводим заголовок к формату: каждое слово с заглавной буквы

# Вывод заголовка и отформатированного заголовка

print("Original Poem Title:", poem\_title)

print("Formatted Poem Title:", poem\_title\_fixed)

**Задание**

1. В переменной строка из первой строки стихотворения Уильяма Карлоса Вильямса «Весенняя буря».

Используйте .split (), чтобы создать список с именем line\_one\_words, который содержит каждое слово в этой строке стихов.

line\_one = "The sky has given over"

Если мы предоставим аргумент для .split (), мы можем указать символ, по которому мы хотим, чтобы наша строка была разделена. Этот аргумент должен быть представлен как сама строка.

Рассмотрим следующий пример:

>>> greatest\_guitarist = "santana"

>>> greatest\_guitarist.split('n')

['sa', 'ta', 'a']

Мы предоставили «n» в качестве аргумента для .split (), поэтому наша строка «santana» была разделена на каждый символ «n» на список из трех строк.

Как вы думаете, что произойдет, если мы разделим одну и ту же строку на «а»?

>>> greatest\_guitarist.split('a')

['s', 'nt', 'n', ' ']

Обратите внимание, что в этом списке есть неожиданная дополнительная строка. Если разделить строку на символ, которым она также заканчивается, в конце списка будет пустая строка.

Вы можете использовать любую строку в качестве аргумента для .split (), что делает его универсальным и мощным инструментом.

# Строка из стихотворения

line\_one = "The sky has given over"

# Используем метод .split() для разделения строки на слова

line\_one\_words = line\_one.split()

# Выводим полученный список слов

print(line\_one\_words)

**Задание**

1. Ваш босс прислал имена авторов, которые он хочет, чтобы вы подготовили для импорта в базу данных. К сожалению, он отправил их в виде длинной строки с именами, разделенными запятыми.

Используя .split () и указанную строку, создайте список с именем author\_names, содержащий имя каждого отдельного автора в виде собственной строки.

authors = "Audre Lorde, Carlos Williams, Gabriela Mistral, Jean Toomer, An Qi, Walt Whitman, Shel Silverstein, Carmen Boullosa, Kamala Suraiyya, Langston Hughes, Adrienne Rich, Nikki Giovanni"

1. Отличная работа, но теперь оказывается, что им не нужны имена поэтов (почему они просто не сказали это в первый раз !?)
2. Создайте еще один список с именем author\_last\_names, который содержит только фамилии поэтов в предоставленной строке.

# Исходная строка с именами авторов

authors = "Audre Lorde, Carlos Williams, Gabriela Mistral, Jean Toomer, An Qi, Walt Whitman, Shel Silverstein, Carmen Boullosa, Kamala Suraiyya, Langston Hughes, Adrienne Rich, Nikki Giovanni"

# 1. Создаем список с именами авторов, используя .split()

author\_names = authors.split(", ")

# Выводим полученный список имен авторов

print("Список имен авторов:")

print(author\_names)

# 2. Убираем имена поэтов из списка (оставляем только фамилии)

# Создаем список фамилий

author\_last\_names = [name.split()[-1] for name in author\_names]

# 3. Выводим полученный список фамилий авторов

print("\nСписок фамилий авторов:")

print(author\_last\_names)

**Задание**

1. Организация прислала вам полный текст стихотворения Уильяма Карлоса Уильямса «Весенняя буря». Они хотят, чтобы вы разбили стихотворение на отдельные строки.

Создайте список с именем spring\_storm\_lines, содержащий строку для каждой строки Spring Storm.

spring\_storm\_text = \

"""The sky has given over

its bitterness.

Out of the dark change

all day long

rain falls and falls

as if it would never end.

Still the snow keeps

its hold on the ground.

But water, water

from a thousand runnels!

It collects swiftly,

dappled with black

cuts a way for itself

through green ice in the gutters.

Drop after drop it falls

from the withered grass-stems

of the overhanging embankment."""

# Полный текст стихотворения

spring\_storm\_text = \

"""The sky has given over

its bitterness.

Out of the dark change

all day long

rain falls and falls

as if it would never end.

Still the snow keeps

its hold on the ground.

But water, water

from a thousand runnels!

It collects swiftly,

dappled with black

cuts a way for itself

through green ice in the gutters.

Drop after drop it falls

from the withered grass-stems

of the overhanging embankment."""

# 1. Разбиваем текст на строки и создаем список

spring\_storm\_lines = spring\_storm\_text.splitlines()

# Выводим полученный список строк

print("Строки стихотворения 'Весенняя буря':")

for line in spring\_storm\_lines:

print(line)

**Задание**

1. Предположим, у нас есть словарь датчиков температуры в доме и текущие значения температуры. Мы только что добавили датчик в «кладовую», он показывает 22 градуса.

Добавьте эту пару в словарь:

sensors =  {"living room": 21, "kitchen": 23, "bedroom": 20}

1. Скопируйте строчку кода ниже и удалите символ # перед определением словаря num\_cameras, который представляет количество камер в каждой области вокруг дома.

Если вы запустите этот код, вы получите сообщение об ошибке:

SyntaxError: invalid syntax

Попробуйте найти и исправить синтаксическую ошибку, чтобы запустить этот код.

#num\_cameras = {"backyard": 6,  "garage": 2 "driveway" 1}

# Изначальный словарь датчиков

sensors = {"living room": 21, "kitchen": 23, "bedroom": 20}

# Добавляем датчик в "кладовую"

sensors["pantry"] = 22 # 'pantry' - это кладовая

# Выводим обновленный словарь

print("Обновленный словарь датчиков:", sensors)

# Исправленный словарь количества камер

num\_cameras = {"backyard": 6, "garage": 2, "driveway": 1}

# Выводим словарь

print("Количество камер в каждой области:", num\_cameras)

**Задание**

1. Создайте словарь переводов, который сопоставляет следующие слова на английском языке с их определениями на синдарине (языке эльфов):

| **English** | **Sindarin** |
| --- | --- |
| mountain | orod |
| bread | bass |
| friend | mellon |
| horse | roch |

# Словарь переводов

translations = {

"mountain": "orod",

"bread": "bass",

"friend": "mellon",

"horse": "roch"

}

# Выводим словарь переводов

print("Словарь переводов:", translations)

**Задание**

1. Создайте пустой словарь с именем animals\_in\_zoo.
2. Гуляя по зоопарку, вы видите 8 зебр. Добавьте «зебры» в animals\_in\_zoo в качестве ключа со значением 8.
3. Добавьте «обезьяны» в animals\_in\_zoo в качестве ключа со значением 12.
4. Покидая зоопарк, вы опечалены тем, что не видели динозавров. Добавьте "динозавров" в animals\_in\_zoo в качестве ключа со значением 0.

Выведите в консоль animals\_in\_zoo.  
  
  
# 1. Создаем пустой словарь

animals\_in\_zoo = {}

# 2. Добавляем 8 зебр

animals\_in\_zoo["зебры"] = 8

# 3. Добавляем 12 обезьян

animals\_in\_zoo["обезьяны"] = 12

# 4. Добавляем 0 динозавров

animals\_in\_zoo["динозавры"] = 0

# 5. Выводим animals\_in\_zoo в консоль

print(animals\_in\_zoo)

**Задание**

1. К словарю user\_ids, представленному ниже, добавьте двух новых пользователей:

* theLooper, с id of 138475
* stringQueen, с id of 85739

>>ser\_ids = {"teraCoder": 9018293, "proProgrammer": 119238}

Выведите в консоль результат  
  
  
# 1. Исходный словарь user\_ids

user\_ids = {"teraCoder": 9018293, "proProgrammer": 119238}

# Добавляем новых пользователей

user\_ids["theLooper"] = 138475

user\_ids["stringQueen"] = 85739

# 2. Выводим результат в консоль

print(user\_ids)

**Задание**

* 1. Добавьте к словарю oscar\_winners  ключ «"Supporting Actress"» и установите значение «Viola Davis».

oscar\_winners = {"Best Picture": "La La Land", "Best Actor": "Casey Affleck", "Best Actress": "Emma Stone", "Animated Feature": "Zootopia"}

Не меняя определения словаря oscar\_winners, измените значение, связанное с ключом «Best Picture», на «Moonlight».  
  
  
  
# Исходный словарь oscar\_winners

oscar\_winners = {

"Best Picture": "La La Land",

"Best Actor": "Casey Affleck",

"Best Actress": "Emma Stone",

"Animated Feature": "Zootopia"

}

# 1. Добавляем новую запись для "Supporting Actress"

oscar\_winners["Supporting Actress"] = "Viola Davis"

# 2. Изменяем значение для "Best Picture"

oscar\_winners["Best Picture"] = "Moonlight"

# Выводим обновленный словарь

print(oscar\_winners)

**Задание**

1. У вас есть два списка, в которых представлены некоторые напитки, продаваемые в кафе, и миллиграммы кофеина в каждом. Сначала создайте переменную с именем zipped\_drinks, которая является итеративным типом, включающим пары между списком напитков и списком кофеина.
2. Создайте словарь с именем drinks\_to\_caffeine, используя генерирование словаря, которое проходит через итератор zipped\_drinks и превращает каждую пару кортежей в элемент ключ: значение.

drinks = ["espresso", "chai", "decaf", "drip"]

caffeine = [64, 40, 0, 120]

# Списки с напитками и содержанием кофеина

drinks = ["espresso", "chai", "decaf", "drip"]

caffeine = [64, 40, 0, 120]

# 1. Создаем переменную zipped\_drinks, которая содержит пары (напиток, кофеин)

zipped\_drinks = zip(drinks, caffeine)

# 2. Создаем словарь drinks\_to\_caffeine с использованием генератора словарей

drinks\_to\_caffeine = {drink: mg\_caffeine for drink, mg\_caffeine in zipped\_drinks}

# Выводим полученный словарь

print(drinks\_to\_caffeine)

**Задание**

1. Мы создаем музыкальный стриминговый сервис. Мы предоставили два списка, в которых представлены песни в библиотеке пользователя и количество воспроизведений каждой песни.

Используя генерирование словаря, создайте словарь под названием plays, который проходит через zip (песни, playcounts) и создает пару song: playcount для каждой песни в песнях и каждого playcount в playcounts.

songs = ["Like a Rolling Stone", "Satisfaction", "Imagine", "What's Going On", "Respect", "Good Vibrations"]

playcounts = [78, 29, 44, 21, 89, 5]

1. Выведите plays.
2. После этого добавьте к plays новую запись. Запись должна быть для песни "Purple Haze", а количество воспроизведений - 1.
3. Этот пользователь заразился лихорадкой Ареты Франклин и прослушал «Respect» 5 раз.
4. Создайте словарь, называемый library, который имеет две пары ключ: значение:

«The Best Songs» со значением plays, (т.е. словарь, который вы создали использовать в качестве значения)

ключ "Sunday Feelings" со значением пустого словаря

1. Выведите library.

# Списки с песнями и количеством воспроизведений

songs = ["Like a Rolling Stone", "Satisfaction", "Imagine", "What's Going On", "Respect", "Good Vibrations"]

playcounts = [78, 29, 44, 21, 89, 5]

# 1. Создаем словарь plays с использованием генерации словаря

plays = {song: count for song, count in zip(songs, playcounts)}

# 2. Выводим plays

print(plays)

# 3. Добавляем новую запись для песни "Purple Haze" с количеством воспроизведений 1

plays["Purple Haze"] = 1

# 4. Увеличиваем количество воспроизведений для "Respect" на 5

plays["Respect"] += 5

# 5. Создаем словарь library

library = {

"The Best Songs": plays,

"Sunday Feelings": {}

}

# 6. Выводим library

print(library)

**Задание**

1. Мы предоставили словарь, в котором элементы стихий соотносятся со знаками зодиака. Выведите в консоль список знаков зодиака, связанных со стихией «земля».

zodiac\_elements = {"water": ["Cancer", "Scorpio", "Pisces"], "fire": ["Aries", "Leo", "Sagittarius"], "earth": ["Taurus", "Virgo", "Capricorn"], "air":["Gemini", "Libra", "Aquarius"]}

1. Выведите в консоль список знаков стихии огня.

# Словарь с элементами и соответствующими знаками зодиака

zodiac\_elements = {

"water": ["Cancer", "Scorpio", "Pisces"],

"fire": ["Aries", "Leo", "Sagittarius"],

"earth": ["Taurus", "Virgo", "Capricorn"],

"air": ["Gemini", "Libra", "Aquarius"]

}

# 1. Выводим знаки зодиака, связанные со стихией «земля»

earth\_signs = zodiac\_elements["earth"]

print("Знаки зодиака стихии Земли:", earth\_signs)

# 2. Выводим знаки зодиака, связанные со стихией огня

fire\_signs = zodiac\_elements["fire"]

print("Знаки зодиака стихии Огня:", fire\_signs)

**Задание**

1. Запустите код. Он должен выдать KeyError! «energy» не существует как один из элементов.
2. Добавьте ключ «energy» к zodiac\_elements. Он должен соответствовать значению «Not a Zodiac element». Запустите код. Устранено ли это KeyError?

zodiac\_elements = {"water": ["Cancer", "Scorpio", "Pisces"], "fire": ["Aries", "Leo", "Sagittarius"], "earth": ["Taurus", "Virgo", "Capricorn"], "air":["Gemini", "Libra", "Aquarius"]}

# Исходный словарь с элементами и соответствующими знаками зодиака

zodiac\_elements = {

"water": ["Cancer", "Scorpio", "Pisces"],

"fire": ["Aries", "Leo", "Sagittarius"],

"earth": ["Taurus", "Virgo", "Capricorn"],

"air": ["Gemini", "Libra", "Aquarius"]

}

# 1. Попробуем получить элементы по ключу "energy"

try:

energy\_signs = zodiac\_elements["energy"]

except KeyError as e:

print(f"KeyError: {e} - ключ 'energy' не существует.")

# 2. Добавляем ключ "energy" с соответствующим значением

zodiac\_elements["energy"] = "Not a Zodiac element"

# Проверяем, устранен ли KeyError

try:

energy\_signs = zodiac\_elements["energy"]

print("Значение для 'energy':", energy\_signs)

except KeyError as e:

print(f"KeyError: {e} - ключ 'energy' не существует.")

**Задание**

1. Используйте блок try/except чтобы попытаться вывести уровень кофеина в матче. Если возникла ошибка KeyError, выведите «Неизвестный уровень кофеина».

caffeine\_level = {"espresso": 64, "chai": 40, "decaf": 0, "drip": 120}

1. Над блоком try добавьте в словарь «matcha» со значением 30

# Создаем словарь с уровнями кофеина

caffeine\_level = {

"espresso": 64,

"chai": 40,

"decaf": 0,

"drip": 120

}

# Добавляем 'matcha' в словарь

caffeine\_level["matcha"] = 30

# Используем блок try/except для получения уровня кофеина в матче

try:

print("Уровень кофеина в матче:", caffeine\_level["matcha"])

except KeyError:

print("Неизвестный уровень кофеина")

**Задание**

1. Используйте .get (), чтобы получить значение идентификатора пользователя teraCoder, со значением по умолчанию 100000, если пользователь не существует. Сохраните его в переменной tc\_id. Выведите tc\_id в консоль.
2. Используйте .get (), чтобы получить значение идентификатора пользователя superStackSmash, со значением по умолчанию 100000, если пользователь не существует. Сохраните его в переменной stack\_id. Вывести stack\_id в консоль.

user\_ids = {"teraCoder": 100019, "pythonGuy": 182921, "samTheJavaMaam": 123112, "lyleLoop": 102931, "keysmithKeith": 129384}

# Словарь идентификаторов пользователей

user\_ids = {

"teraCoder": 100019,

"pythonGuy": 182921,

"samTheJavaMaam": 123112,

"lyleLoop": 102931,

"keysmithKeith": 129384

}

# Получаем идентификатор пользователя teraCoder с использованием .get() и значением по умолчанию

tc\_id = user\_ids.get("teraCoder", 100000)

print("tc\_id:", tc\_id)

# Получаем идентификатор пользователя superStackSmash с использованием .get() и значением по умолчанию

stack\_id = user\_ids.get("superStackSmash", 100000)

print("stack\_id:", stack\_id)

**Задание**

1. Создайте переменную с именем users и назначьте ей объект dict\_keys для всех ключей словаря user\_ids.

user\_ids = {"teraCoder": 100019, "pythonGuy": 182921, "samTheJavaMaam": 123112, "lyleLoop": 102931, "keysmithKeith": 129384}

num\_exercises = {"functions": 10, "syntax": 13, "control flow": 15, "loops": 22, "lists": 19, "classes": 18, "dictionaries": 18}

1. Создайте переменную с именем classes и назначьте ее объектом dict\_keys для всех ключей словаря num\_exercises.
2. Вывести users в консоль.

Вывести lessons на консоли.  
  
  
  
# Словарь идентификаторов пользователей

user\_ids = {

"teraCoder": 100019,

"pythonGuy": 182921,

"samTheJavaMaam": 123112,

"lyleLoop": 102931,

"keysmithKeith": 129384

}

# Словарь количества упражнений

num\_exercises = {

"functions": 10,

"syntax": 13,

"control flow": 15,

"loops": 22,

"lists": 19,

"classes": 18,

"dictionaries": 18

}

# Создаем переменную users, которая содержит все ключи из user\_ids

users = user\_ids.keys()

# Создаем переменную classes, которая содержит все ключи из num\_exercises

classes = num\_exercises.keys()

# Выводим users в консоль

print("Users:", users)

# Выводим classes в консоль

print("Classes:", classes)

**Задание**

1. Создайте переменную с именем total\_exercises и установите ее равной 0.
2. Просмотрите значения в списке num\_exercises и добавьте каждое значение в переменную total\_exercises.
3. Выведите в консоль переменную total\_exercises.

>>num\_exercises = {"functions": 10, "syntax": 13, "control flow": 15, "loops": 22, "lists": 19, "classes": 18, "dictionaries": 18}

# Словарь количества упражнений

num\_exercises = {

"functions": 10,

"syntax": 13,

"control flow": 15,

"loops": 22,

"lists": 19,

"classes": 18,

"dictionaries": 18

}

# Создаем переменную total\_exercises и устанавливаем ее равной 0

total\_exercises = 0

# Проходим по значениям в num\_exercises и добавляем каждое значение к total\_exercises

for exercises in num\_exercises.values():

total\_exercises += exercises

# Выводим значение total\_exercises в консоль

print("Total exercises:", total\_exercises)

**Задание**

1. Мы предоставили колоду карт Таро. Вы собираетесь вытащить из колоды три карты своего прошлого, настоящего и будущего.

tarot = { 1:  "The Magician", 2:  "The High Priestess", 3:  "The Empress", 4: "The Emperor", 5: "The Hierophant", 6:  "The Lovers", 7:  "The Chariot", 8: "Strength", 9:  "The Hermit", 10: "Wheel of Fortune", 11: "Justice", 12:  "The Hanged Man", 13: "Death", 14:  "Temperance", 15: "The Devil", 16:  "The Tower", 17:  "The Star", 18: "The Moon", 19: "The Sun", 20:  "Judgement", 21:  "The World", 22: "The Fool"}

Создайте пустой словарь под названием spread.

1. Первая карта, которую вы берете, — это карта 13. Извлеките значение, присвоенное ключу 13, из словаря Таро и назначьте его как значение к ключу «прошлое».
2. Вторая карта, которую вы берете, — это карта 22. Извлеките значение, присвоенное ключу 22, из словаря Таро и назначьте его как значение ключу «настоящеее».
3. Третья карта, которую вы берете, — это карта 10. Извлеките значение, присвоенное ключу 10, из словаря Таро и назначьте его как значение к ключу «будущее»

# Словарь карт Таро

tarot = {

1: "The Magician",

2: "The High Priestess",

3: "The Empress",

4: "The Emperor",

5: "The Hierophant",

6: "The Lovers",

7: "The Chariot",

8: "Strength",

9: "The Hermit",

10: "Wheel of Fortune",

11: "Justice",

12: "The Hanged Man",

13: "Death",

14: "Temperance",

15: "The Devil",

16: "The Tower",

17: "The Star",

18: "The Moon",

19: "The Sun",

20: "Judgement",

21: "The World",

22: "The Fool"

}

# Создаем пустой словарь для расклада

spread = {}

# Извлекаем карты и назначаем их соответствующим ключам

spread["past"] = tarot[13] # Карта 13

spread["present"] = tarot[22] # Карта 22

spread["future"] = tarot[10] # Карта 10

# Выводим результат расклада

print(spread)

\***Итоговое задание\***

1. Создайте файл csv, содержащий следующие сведения:

|  |  |  |  |
| --- | --- | --- | --- |
| ФИО | Должность | Дата найма | Оклад |
| Иванов Иван Иванович | Менеджер | 22.10.2013 | 250 000 |
| Сорокина Екатерина Матвеевна | Аналитик | 12.03.2020 | 75 000 |
| Струков Иван Сергеевич | Старший программист | 23.04.2012 | 150 000 |
| Корнеева Анна Игоревна | Ведущий программист | 22.02.2015 | 120 000 |
| Старчиков Сергей Анатольевич | Младший программист | 12.11.2021 | 50 000 |
| Бутенко Артем Андреевич | Архитектор | 12.02.2010 | 200 000 |
| Савченко Алина Сергеевна | Старший аналитик | 13.04.2016 | 100 000 |

1. Начальство решило выплачивать премии ко дню программиста в сентябре (13 сентября – день программиста). Премия будет составлять 3% от оклада каждому программисту. Реализуйте функцию, которая будет рассчитывать премию.
2. Начальство решило выплачивать премии к 8 марта всем сотрудницам, а к 23 февраля сотрудникам, равные 2000. Реализуйте функцию, которая будет рассчитывать премию.
3. Было решено провести индексацию зарплат сотрудников. Сотрудникам, которые отработали в компании более 10 лет, индексация будет равна 7% оклада, остальным – 5%.
4. Необходимо составить график отпусков сотрудников. Для этого необходимо в отдельный список занести тех сотрудников, которые отработали в компании более 6 месяцев. Реализовать функцию.
5. Реализуйте функции записи в csv и json.

import csv

# Данные о сотрудниках

employees = [

["ФИО", "Должность", "Дата найма", "Оклад"],

["Иванов Иван Иванович", "Менеджер", "22.10.2013", "250000"],

["Сорокина Екатерина Матвеевна", "Аналитик", "12.03.2020", "75000"],

["Струков Иван Сергеевич", "Старший программист", "23.04.2012", "150000"],

["Корнеева Анна Игоревна", "Ведущий программист", "22.02.2015", "120000"],

["Старчиков Сергей Анатольевич", "Младший программист", "12.11.2021", "50000"],

["Бутенко Артем Андреевич", "Архитектор", "12.02.2010", "200000"],

["Савченко Алина Сергеевна", "Старший аналитик", "13.04.2016", "100000"],

]

# Создание CSV файла

with open("employees.csv", mode="w", newline="", encoding="utf-8") as file:

writer = csv.writer(file)

writer.writerows(employees)

print("CSV файл создан.")

from datetime import datetime

# Функция для расчета премии программистам

def calculate\_programmer\_bonus(salary):

return salary \* 0.03 # 3% от оклада

# Функция для расчета премии ко Дню женщин и 23 февраля

def calculate\_gender\_bonus(position, salary, hire\_date):

if "программист" in position.lower():

return 0

hire\_date\_obj = datetime.strptime(hire\_date, "%d.%m.%Y")

current\_date = datetime.now()

if hire\_date\_obj.year < current\_date.year - 10: # Более 10 лет

return 2000

elif position in ["Менеджер", "Аналитик", "Архитектор", "Старший аналитик"]:

return 2000

return 0

# Функция для индексации зарплат

def index\_salary(hire\_date, salary):

hire\_date\_obj = datetime.strptime(hire\_date, "%d.%m.%Y")

years = (datetime.now() - hire\_date\_obj).days / 365

if years > 10:

return salary \* 1.07 # 7%

else:

return salary \* 1.05 # 5%

# Функция для формирования списка отпусков

def list\_employees\_on\_leave(hire\_date):

hire\_date\_obj = datetime.strptime(hire\_date, "%d.%m.%Y")

if (datetime.now() - hire\_date\_obj).days / 30 > 6:

return True

return False

import json

# Функция записи данных в CSV

def write\_to\_csv(file\_name, data):

with open(file\_name, mode="w", newline="", encoding="utf-8") as file:

writer = csv.writer(file)

writer.writerows(data)

# Функция записи данных в JSON

def write\_to\_json(file\_name, data):

with open(file\_name, mode="w", encoding="utf-8") as file:

json.dump(data, file, ensure\_ascii=False, indent=4)

# Сбор данных для JSON

employee\_data = []

for employee in employees[1:]: # Пропускаем заголовок

fio, position, hire\_date, salary = employee

salary = int(salary.replace(" ", ""))

# Рассчитываем премии

programmer\_bonus = calculate\_programmer\_bonus(salary) if "программист" in position.lower() else 0

gender\_bonus = calculate\_gender\_bonus(position, salary, hire\_date)

# Индексация зарплат

new\_salary = index\_salary(hire\_date, salary)

# Проверка на отпуск

on\_leave = list\_employees\_on\_leave(hire\_date)

employee\_data.append({

"ФИО": fio,

"Должность": position,

"Дата найма": hire\_date,

"Оклад": salary,

"Премия программистов": programmer\_bonus,

"Премия": gender\_bonus,

"Индексация": new\_salary,

"В отпуске": on\_leave

})

# Запись в CSV и JSON

write\_to\_csv("employees\_with\_bonuses.csv", employee\_data)

write\_to\_json("employees\_data.json", employee\_data)

print("Данные записаны в CSV и JSON.")