МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

ГЛАВНОЕ УПРАВЛЕНИЕ ПО ОБРАЗОВАНИЮ

МОГИЛЕВСКОГО ОБЛАСТНОГО ИСПОЛНИТЕЛЬНОГО КОМИТЕТА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«МОГИЛЕВСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

|  |  |
| --- | --- |
| Специальность | 2– 40 01 01 |
| Учебная группа | ПО-455 |
|  |  |

Учебная дисциплина Инструментальное

программное обеспечение

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5**

**РАЗРАБОТКА, ОТЛАДКА И ИСПЫТАНИЕ АЛГОРИТМОВ И ПРОГРАММ РАБОТЫ СО СПИСКАМИ**

Выполнил Савич А.О.

Проверил Денисовец Д. А.

2022

1. **Цель работы**

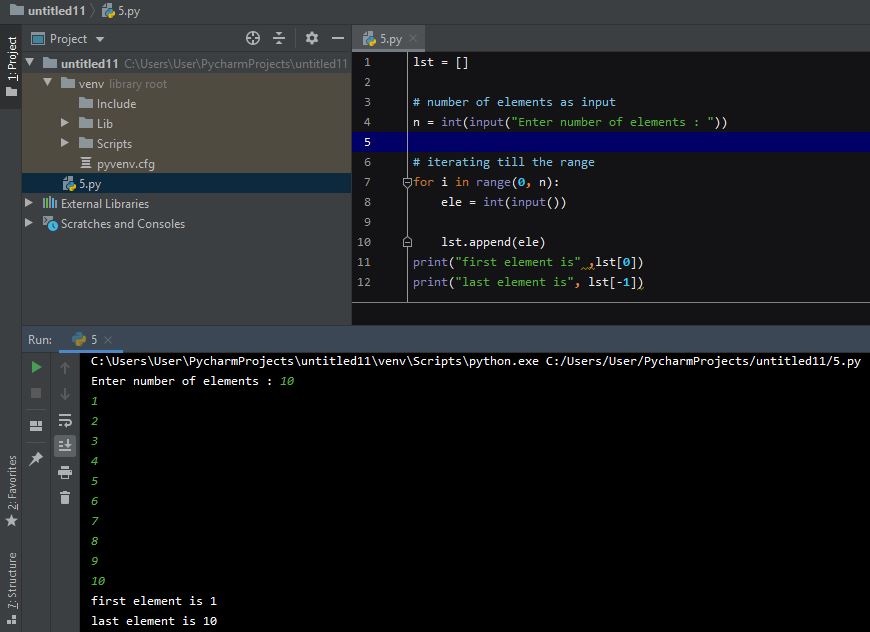
Формирование умений разрабатывать, выполнять отладку и испытание алгоритмов и программ работы со списками

**2 Индивидуальное задание**

# Вариант 8

Разработайте алгоритм и напишите программу решения задачи на языке программирования Python в соответствии с заданным вариантом.

Выведите первый и последний элемент списка.



**3 Контрольные вопросы**

1 Опишите способы создания списков в языке Python.

Для работы с наборами данных Python предоставляет такие встроенные типы как списки, кортежи и словари.

Список (list) представляет тип данных, который хранит набор или последовательность элементов. Во многих языках программирования есть аналогичная структура данных, которая называется массив.

### Создание списка

Для создания списка применяются квадратные скобки **[]**, внутри которых через запятую перечисляются элементы списка. Например, определим список чисел:

numbers = [1, 2, 3, 4, 5]

Подобным образом можно определять списки с данными других типов, например, определим список строк:

people = ["Tom", "Sam", "Bob"]

Также для создания списка можно использовать функцию-конструктор **list()**:

numbers1 = []

numbers2 = list()

Оба этих определения списка аналогичны - они создают пустой список.

Список необязательно должен содержать только однотипные объекты. Мы можем поместить в один и тот же список одновременно строки, числа, объекты других типов данных:

objects = [1, 2.6, "Hello", True]

Для проверки элементов списка можно использовать стандартную функцию print, которая выводит содержимое списка в удобочитаемом виде:

numbers = [1, 2, 3, 4, 5]

people = ["Tom", "Sam", "Bob"]

print(numbers) # [1, 2, 3, 4, 5]

print(people) # ["Tom", "Sam", "Bob"]

Конструктор list может принимать набор значений, на основе которых создается список:

numbers1 = [1, 2, 3, 4, 5]

numbers2 = list(numbers1)

print(numbers2) # [1, 2, 3, 4, 5]

letters = list("Hello")

print(letters) # ['H', 'e', 'l', 'l', 'o']

Если необходимо создать список, в котором повторяется одно и то же значение несколько раз, то можно использовать символ звездочки \*, то есть фактически применить операцию умножения к уже существующему списку:

numbers = [5] \* 6 # 6 раз повторяем 5

print(numbers) # [5, 5, 5, 5, 5, 5]

people = ["Tom"] \* 3 # 3 раза повторяем "Tom"

print(people) # ["Tom", "Tom", "Tom"]

students = ["Bob", "Sam"] \* 2 # 2 раза повторяем "Bob", "Sam"

print(students) # ["Bob", "Sam", "Bob", "Sam"]

### Обращение к элементам списка

Для обращения к элементам списка надо использовать индексы, которые представляют номер элемента в списка. Индексы начинаются с нуля. То есть первый элемент будет иметь индекс 0, второй элемент - индекс 1 и так далее. Для обращения к элементам с конца можно использовать отрицательные индексы, начиная с -1. То есть у последнего элемента будет индекс -1, у предпоследнего - -2 и так далее.

people = ["Tom", "Sam", "Bob"]

# получение элементов с начала списка

print(people[0]) # Tom

print(people[1]) # Sam

print(people[2]) # Bob

# получение элементов с конца списка

print(people[-2]) # Sam

print(people[-1]) # Bob

print(people[-3]) # Tom

Для изменения элемента списка достаточно присвоить ему новое значение:

people = ["Tom", "Sam", "Bob"]

people[1] = "Mike" # изменение второго элемента

print(people[1]) # Mike

print(people) # ["Tom", "Mike", "Bob"]

### Разложение списка

Python позволяет разложить список на отдельные элементы:

people = ["Tom", "Bob", "Sam"]

tom, bob, sam = people

print(tom) # Tom

print(bob) # Bob

print(sam) # Sam

В данном случае переменным tom, bob и sam последовательно присваиваются элементы из списка people. Однако следует учитывать, что количество переменных должно быть равно числу элементов присваиваемого списка.

### Перебор элементов

Для перебора элементов можно использовать как цикл for, так и цикл while.

Перебор с помощью цикла **for**:

people = ["Tom", "Sam", "Bob"]

for person in people:

print(person)

Здесь будет производиться перебор списка people, и каждый его элемент будет помещаться в переменную person.

Перебор также можно сделать с помощью цикла **while**:

people = ["Tom", "Sam", "Bob"]

i = 0

while i < len(people):

print(people[i]) # применяем индекс для получения элемента

i += 1

Для перебора с помощью функции **len()** получаем длину списка. С помощью счетчика i выводит по элементу, пока значение счетчика не станет равно длине списка.

### Сравнение списков

Два списка считаются равными, если они содержат один и тот же набор элементов:

numbers1 = [1, 2, 3, 4, 5]

numbers2 = list([1, 2, 3, 4, 5])

if numbers1 == numbers2:

print("numbers1 equal to numbers2")

else:

print("numbers1 is not equal to numbers2")

В данном случае оба списка будут равны.

### Методы и функции по работе со списками

Для управления элементами списки имеют целый ряд методов. Некоторые из них:

* **append(item)**: добавляет элемент item в конец списка
* **insert(index, item)**: добавляет элемент item в список по индексу index
* **extends(items)**: добавляет набор элементов items в конец списка
* **remove(item)**: удаляет элемент item. Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение ValueError
* **clear()**: удаление всех элементов из списка
* **index(item)**: возвращает индекс элемента item. Если элемент не найден, генерирует исключение ValueError
* **pop([index])**: удаляет и возвращает элемент по индексу index. Если индекс не передан, то просто удаляет последний элемент.
* **count(item)**: возвращает количество вхождений элемента item в список
* **sort([key])**: сортирует элементы. По умолчанию сортирует по возрастанию. Но с помощью параметра key мы можем передать функцию сортировки.
* **reverse()**: расставляет все элементы в списке в обратном порядке
* **copy()**: копирует список

Кроме того, Python предоставляет ряд встроенных функций для работы со списками:

* **len(list)**: возвращает длину списка
* **sorted(list, [key])**: возвращает отсортированный список
* **min(list)**: возвращает наименьший элемент списка
* **max(list)**: возвращает наибольший элемент списка

### Добавление и удаление элементов

Для добавления элемента применяются методы append(), extends и insert, а для удаления - методы remove(), pop() и clear().

Использование методов:

people = ["Tom", "Bob"]

# добавляем в конец списка

people.append("Alice") # ["Tom", "Bob", "Alice"]

# добавляем на вторую позицию

people.insert(1, "Bill") # ["Tom", "Bill", "Bob", "Alice"]

# добавляем набор элементов ["Mike", "Sam"]

people.extend(["Mike", "Sam"]) # ["Tom", "Bill", "Bob", "Alice", "Mike", "Sam"]

# получаем индекс элемента

index\_of\_tom = people.index("Tom")

# удаляем по этому индексу

removed\_item = people.pop(index\_of\_tom) # ["Bill", "Bob", "Alice", "Mike", "Sam"]

# удаляем последний элемент

last\_item = people.pop() # ["Bill", "Bob", "Alice", "Mike"]

# удаляем элемент "Alice"

people.remove("Alice") # ["Bill", "Bob", "Mike"]

print(people) # ["Bill", "Bob", "Mike"]

# удаляем все элементы

people.clear()

print(people) # []

### Проверка наличия элемента

Если определенный элемент не найден, то методы remove и index генерируют исключение. Чтобы избежать подобной ситуации, перед операцией с элементом можно проверять его наличие с помощью ключевого слова **in**:

people = ["Tom", "Bob", "Alice", "Sam"]

if "Alice" in people:

people.remove("Alice")

print(people) # ["Tom", "Bob", "Sam"]

Выражение if "Alice" in people возвращает True, если элемент "Alice" имеется в списке people. Поэтому конструкция if "Alice" in people может выполнить последующий блок инструкций в зависимости от наличия элемента в списке.

### Удаление с помощью del

Python также поддерживает еще один способ удаления элементов списка - с помощью оператора **del**. В качестве параметра этому оператору передается удаляемый элемент или набор элементов:

people = ["Tom", "Bob", "Alice", "Sam", "Bill", "Kate", "Mike"]

del people[1] # удаляем второй элемент

print(people) # ["Tom", "Alice", "Sam", "Bill", "Kate", "Mike"]

del people[:3] # удаляем по четвертый элемент не включая

print(people) # ["Bill", "Kate", "Mike"]

del people[1:] # удаляем со второго элемента

print(people) # ["Bill"]

### Подсчет вхождений

Если необходимо узнать, сколько раз в списке присутствует тот или иной элемент, то можно применить метод count():

people = ["Tom", "Bob", "Alice", "Tom", "Bill", "Tom"]

people\_count = people.count("Tom")

print(people\_count) # 3

### Сортировка

Для сортировки по возрастанию применяется метод **sort()**:

people = ["Tom", "Bob", "Alice", "Sam", "Bill"]

people.sort()

print(people) # ["Alice", "Bill", "Bob", "Sam", "Tom"]

Если необходимо отсортировать данные в обратном порядке, то мы можем после сортировки применить метод reverse():

people = ["Tom", "Bob", "Alice", "Sam", "Bill"]

people.sort()

people.reverse()

print(people) # ["Tom", "Sam", "Bob", "Bill", "Alice"]

При сортировке фактически сравниваются два объекта, и который из них "меньше", ставится перед тем, который "больше". Понятия "больше" и "меньше" довольно условны. И если для чисел все просто - числа расставляются в порядке возрастания, то для строк и других объектов ситуация сложнее. В частности, строки оцениваются по первым символам. Если первые символы равны, оцениваются вторые символы и так далее. При чем цифровой символ считается "меньше", чем алфавитный заглавный символ, а заглавный символ считается меньше, чем строчный.

Таким образом, если в списке сочетаются строки с верхним и нижним регистром, то мы можем получить не совсем корректные результаты, так как для нас строка "bob" должна стоять до строки "Tom". И чтобы изменить стандартное поведение сортировки, мы можем передать в метод sort() в качестве параметра функцию:

people = ["Tom", "bob", "alice", "Sam", "Bill"]

people.sort() # стандартная сортировка

print(people) # ["Bill", "Sam", "Tom", "alice", "bob"]

people.sort(key=str.lower) # сортировка без учета регистра

print(people) # ["alice", "Bill", "bob", "Sam", "Tom"]

Кроме метода sort мы можем использовать встроенную функцию **sorted**, которая имеет две формы:

* sorted(list): сортирует список list
* sorted(list, key): сортирует список list, применяя к элементам функцию key

people = ["Tom", "bob", "alice", "Sam", "Bill"]

sorted\_people = sorted(people, key=str.lower)

print(sorted\_people) # ["alice", "Bill", "bob", "Sam", "Tom"]

При использовании этой функции следует учитывать, что эта функция не изменяет сортируемый список, а все отсортированные элементы она помещает в новый список, который возвращается в качестве результата.

### Минимальное и максимальное значения

Встроенный функции Python min() и max() позволяют найти минимальное и максимальное значения соответственно:

numbers = [9, 21, 12, 1, 3, 15, 18]

print(min(numbers)) # 1

print(max(numbers)) # 21

### Копирование списков

При копировании списков следует учитывать, что списки представляют изменяемый (mutable) тип, поэтому если обе переменных будут указывать на один и тот же список, то изменение одной переменной, затронет и другую переменную:

people1 = ["Tom", "Bob", "Alice"]

people2 = people1

people2.append("Sam") # добавляем элемент во второй список

# people1 и people2 указывают на один и тот же список

print(people1) # ["Tom", "Bob", "Alice", "Sam"]

print(people2) # ["Tom", "Bob", "Alice", "Sam"]

Это так называемое "поверхностное копирование" (shallow copy). И, как правило, такое поведение нежелательное. И чтобы происходило копирование элементов, но при этом переменные указывали на разные списки, необходимо выполнить глубокое копирование (deep copy). Для этого можно использовать метод **copy()**:

people1 = ["Tom", "Bob", "Alice"]

people2 = people1.copy() # копируем элементы из people1 в people2

people2.append("Sam") # добавляем элемент ТОЛЬКО во второй список

# people1 и people2 указывают на разные списки

print(people1) # ["Tom", "Bob", "Alice"]

print(people2) # ["Tom", "Bob", "Alice", "Sam"]

### Копирование части списка

Если необходимо скопировать не весь список, а только его какую-то определенную часть, то мы можем применять специальный синтаксис. который может принимать следующие формы:

* list[:end]: через параметр end передается индекс элемента, до которого нужно копировать список
* list[start:end]: параметр start указывает на индекс элемента, начиная с которого надо скопировать элементы
* list[start:end:step]: параметр step указывает на шаг, через который будут копироваться элементы из списка. По умолчанию этот параметр равен 1.

people = ["Tom", "Bob", "Alice", "Sam", "Tim", "Bill"]

slice\_people1 = people[:3] # с 0 по 3

print(slice\_people1) # ["Tom", "Bob", "Alice"]

slice\_people2 = people[1:3] # с 1 по 3

print(slice\_people2) # ["Bob", "Alice"]

slice\_people3 = people[1:6:2] # с 1 по 6 с шагом 2

print(slice\_people3) # ["Bob", "Sam", "Bill"]

### Соединение списков

Для объединения списков применяется операция сложения (+):

people1 = ["Tom", "Bob", "Alice"]

people2 = ["Tom", "Sam", "Tim", "Bill"]

people3 = people1 + people2

print(people3) # ["Tom", "Bob", "Alice", "Tom", "Sam", "Tim", "Bill"]

### Списки списков

Списки кроме стандартных данных типа строк, чисел, также могут содержать другие списки. Подобные списки можно ассоциировать с таблицами, где вложенные списки выполняют роль строк. Например:

people = [

["Tom", 29],

["Alice", 33],

["Bob", 27]

]

print(people[0]) # ["Tom", 29]

print(people[0][0]) # Tom

print(people[0][1]) # 29

Чтобы обратиться к элементу вложенного списка, необходимо использовать пару индексов: people[0][1] - обращение ко второму элементу первого вложенного списка.

Добавление, удаление и изменение общего списка, а также вложенных списков аналогично тому, как это делается с обычными (одномерными) списками

1. Перечислите операции над списками в языке Python.

Для работы со списками используются следующие основные операции:

**+** – конкатенация списков;

**\*** – дублирование списка;

len – определение длины списка;

del – удаление элемента с списка;

присваивание по индексу;

использование среза для доступа к элементам списка и изменения списка.

3 Охарактеризуйте добавление и удаление элементов списка в языке Python.

**При занесении новых данных в список его размер автоматически увеличивается на нужное количество ячеек**. Для того чтобы в Python добавить в список элемент стоит воспользоваться методом append, передав ему в качестве аргумента добавляемый объект. В результате выполнения этой функции, элемент будет добавлен в конец.

При необходимости **любой из объектов можно размещать по заданному индексу в списке**. Для этого используется метод с именем insert. Вызвав его для существующего списка, нужно передать в качестве аргументов желаемый номер элемента и сам объект.

4 Приведите синтаксис сортировки списков в языке Python.

Встроенные функции платформы позволяют не только добавлять новые данные, но и выборочно избавляться от них. Для того, чтобы удалить элемент из списка Python существуют функции remove и pop.**В случае с первым методом происходит удаление первого по индексу объекта с выбранным значением**.

**Вывод:** Научился разрабатывать, выполнять отладку и испытание алгоритмов и программ работы со списками