Введение

Алгоритмические языки и программирование играют важную роль в современном мире, и они являются основой для создания компьютерных программ и приложений. Введение в эту область включает в себя следующие ключевые концепции и темы:

Что такое алгоритм?: Алгоритм - это последовательность инструкций, которая описывает, как выполнить конкретную задачу. Он является базовым элементом программирования. Важно понимать, что алгоритмы могут быть выражены на разных языках программирования.

Языки программирования: Язык программирования - это формальный способ записи алгоритмов для выполнения компьютером. Некоторые популярные языки программирования включают Python, Java, C++, JavaScript, и многие другие. Каждый из них имеет свои особенности и предназначен для решения различных задач.

Синтаксис и семантика: Каждый язык программирования имеет свой собственный синтаксис (правила написания кода) и семантику (смысл и интерпретацию кода). Понимание этих аспектов является важной частью обучения программированию.

Парадигмы программирования: Существует несколько парадигм программирования, такие как императивное, объектно-ориентированное, функциональное, и другие. Каждая из них предоставляет разные способы организации кода и решения задач.

Базовые структуры данных: Для обработки данных и хранения информации программисты используют различные структуры данных, такие как массивы, списки, словари, стеки и очереди.

Управление потоком: Управление потоком включает в себя конструкции, такие как условные операторы (if-else), циклы (for, while), и операторы перехода. Они позволяют управлять тем, как программа выполняется в зависимости от различных условий.

Отладка и тестирование: Важной частью программирования является отладка, которая включает в себя поиск и устранение ошибок в коде, а также тестирование для проверки корректности работы программы.

Разработка алгоритмов: Создание эффективных алгоритмов для решения задач - ключевой навык программиста. Это включает в себя анализ задачи, выбор подходящих структур данных и разработку эффективных алгоритмов.

Работа с библиотеками и фреймворками: В большинстве случаев программисты используют готовые библиотеки и фреймворки для ускорения разработки. Понимание, как использовать эти ресурсы, также важно.

Практика и проекты: Практика является наилучшим способом усвоить навыки программирования. Работа над реальными проектами позволяет применять полученные знания на практике.

Введение в алгоритмические языки и программирование может быть начато с изучения конкретного языка программирования, а затем переходить к более общим концепциям и навыкам.

Методическая разработка лекций по дисциплине (Краткий курс лекций, презентации)

Глава № 4. Коллекции данных в Python.

Понимание различных коллекций данных в Python и их использование являются важной частью разработки на этом языке. В этой главе введения мы рассмотрим основные типы коллекций, которые часто используются в Python.

Строки (Strings): Строки представляют собой последовательности символов, их можно определить с использованием одинарных или двойных кавычек. Строки используются для хранения текстовых данных. В Python строки неизменяемы, что означает, что после создания строки ее нельзя изменить. Работа со строками включает в себя конкатенацию, индексацию, срезы и множество методов для манипуляции текстовыми данными.

Множества (Sets): Множества представляют собой коллекции уникальных элементов без упорядочения. Множества в Python позволяют выполнять операции объединения, пересечения и разности. Они полезны, когда вам нужно убедиться, что элементы не дублируются в коллекции.

Списки (Lists): Списки представляют упорядоченные коллекции элементов. В отличие от множеств, в списках элементы могут повторяться. Списки изменяемы, и вы можете добавлять, удалять и изменять элементы. Списки широко используются в Python.

Кортежи (Tuples): Кортежи похожи на списки, но они неизменяемы. Кортежи используются для хранения неизменяемых наборов данных и часто используются в случаях, когда необходима надежность и стабильность данных.

Словари (Dictionaries): Словари представляют собой коллекции пар "ключ-значение". Они позволяют быстро находить и извлекать данные по ключу. Словари очень полезны для хранения структурированных данных и решения задач, связанных с поиском.

Вложенные последовательности: В Python вы можете вкладывать одни коллекции внутрь других. Например, в список или кортеж можно вложить другие списки, кортежи или словари. Это позволяет вам представлять структурированные данные с более высокой сложностью.

В этой главе введения мы рассмотрели основные типы коллекций в Python. Далее, в более практических уроках, вы узнаете, как эффективно работать с каждым из этих типов данных, а также как комбинировать и вкладывать их для решения разнообразных задач.

Тема №13.Работа со строками: основные понятия, операции с ними. Методы строк.Форматирование строк. Базовые алгоритмы обработки строк.

Работа с строками в Python - это фундаментальная часть программирования, поскольку строки часто используются для хранения и манипулирования текстовой информацией. Вот основные понятия и операции для работы со строками в Python:

Основные понятия и операции:

1.Строка (String): Строка - это последовательность символов. В Python строки могут быть определены с использованием одинарных (') или двойных (") кавычек.

Примеры:

```
string1 = "Привет, мир!"
string2 = 'Python - отличный язык программирования.'
```

2.Конкатенация: Конкатенация строк - это операция объединения строк с использованием оператора +.

Пример:

```
privetstviye = "Привет"
name = "Чынара"
full_privetstviye = privetstviye + " " + name
print(full_privetstviye)
```

результат:

```
Привет Чынара
```

3.Индексация и срезы: Строки индексируются, начиная с 0. Вы можете получать доступ к отдельным символам или подстрокам, используя квадратные скобки []. Также вы можете использовать срезы для извлечения подстроки.

Примеры:

```
text = "Python"
first_char = text[0]   # Первая буква 'P'
substring = text[2:4]   # Подстрока 'th'
```

4.Длина строки: Для определения длины строки используется функция len().

Пример:

```
text = "Python"
length = len(text)   # Длина строки - 6
```

Методы строк

Python предоставляет множество методов для работы со строками. Некоторые из наиболее часто используемых методов включают:

1.split(): в Python используется для разделения строки на подстроки (токены) на основе заданного разделителя (по умолчанию разделитель - пробел). Этот метод очень полезен, когда вам нужно разбить текст на отдельные части для дальнейшей обработки.

Вот синтаксис метода split():

```
string.split([separator[, maxsplit]])
```

separator (необязательный): Это строка, которая определяет, какие символы будут использоваться в качестве разделителей. По умолчанию используется пробел. Может быть любой строкой, например, запятой ",", тире "-", точкой с запятой ";", и т. д.

maxsplit (необязательный): Этот параметр определяет максимальное количество разделений. Если он указан, строка будет разделена не более maxsplit раз. Если maxsplit не указан, строка будет разделена на все возможные подстроки.

Примеры использования метода split():

```
text = "Привет, мир! Как дела?"
words = text.split()  # Разделяет строку по пробелам, по умолчанию
print(words)
# Вывод: ['Привет,', 'мир!', 'Как', 'дела?']

text = "Чынара,Ак-ордо,30,Бишкек"
words = text.split(",")  # Разделение строки по запятой
print(words)
# Вывод: ['Чынара', 'Ак-ордо', '30', 'Бишкек']

text = "Это просто предложение."
words = text.split(" ", 2)  # Разделение строки на максимум 2 части
print(words)
# Вывод: ['Это', 'просто', 'предложение.']
```

Метод split() возвращает список подстрок, полученных после разделения исходной строки. Эти подстроки можно затем использовать для выполнения различных операций обработки текста.

2.strip(): в Python используется для удаления лишних пробелов (или других указанных символов) в начале и конце строки. Он полезен, когда вам нужно очистить строку от пробелов, переносов строк, табуляций и других символов, которые могут быть нежелательными.

Вот синтаксис метода strip():

```
string.strip([characters])
```

characters (необязательный): Это строка, которая содержит символы, которые вы хотите удалить с начала и конца строки. По умолчанию удаляются пробелы. Вы можете указать свой набор символов для удаления.

Примеры использования метода strip():

```
text = "   Привет, мир!   "
text1 = text.strip()
print(text1)
# Вывод: 'Привет, мир!'

text = "###Это заголовок###"
text2 = text.strip("#")
print(text2)
# Вывод: 'Это заголовок'

text = "\n\n\nТекст с переносами строк\n\n\n"
text3 = text.strip("\n")
print(text3)
# Вывод: 'Текст с переносами строк'
```

Метод strip() возвращает новую строку, которая представляет собой исходную строку после удаления указанных символов в начале и конце. Исходная строка не изменяется. Вы можете использовать этот метод для очистки ввода данных или для предварительной обработки строк перед их анализом.

3.upper() и lower(): в Python используются для изменения регистра букв в строке. Вот как они работают:

upper(): Этот метод преобразует все буквы в строке в верхний регистр (заглавные буквы). Все символы, которые не являются буквами, остаются без изменений.

Пример использования upper():

```
text = "Привет, мир!"
text1 = text.upper()
print(text1)
# Вывод: 'ПРИВЕТ, МИР!'
```

lower(): Этот метод преобразует все буквы в строке в нижний регистр (строчные буквы). Все символы, которые не являются буквами, остаются без изменений.

Пример использования lower():

```
text = "Привет, Мир!"
text1 = text.lower()
print(text1)
# Вывод: 'привет, мир!'
```

Эти методы полезны, когда вам нужно стандартизировать регистр символов в строках. Например, вы можете использовать upper() или lower() для сравнения строк без учета регистра или для преобразования введенных данных в желаемый регистр перед их обработкой.

4.replace(): в Python используется для замены подстроки в строке другой подстрокой. Этот метод полезен, когда вам нужно выполнить операцию замены в текстовых данных.

Вот синтаксис метода replace():

```
string.replace(old, new[, count])
```

old:- это подстрока, которую вы хотите заменить.

new: -это подстрока, на которую вы хотите заменить old.

count (необязательный):-этот параметр определяет, сколько раз нужно выполнить замену. Если он не указан, то заменятся все вхождения old.

Примеры использования replace():

```
text = "Привет, мир!"
new_text = text.replace("мир", "Python")
print(new_text)
# Вывод: 'Привет, Python!'

text = "Это длинное предложение. Длинное."
text1 = text.replace("Длинное", "Короткое",1)
print(text1)
#Вывод: 'Это длинное предложение. Короткое.'

text = "1,000,000"
text1 = text.replace(",", "")
print(text1)
# Вывод: '1000000'
```

Метод replace() возвращает новую строку, в которой все вхождения old были заменены на new. Исходная строка остается без изменений. Вы также можете использовать параметр count, чтобы ограничить количество замен.

5.startswith() и endswith(): в Python используются для проверки начала и конца строки соответственно. Они позволяют вам определить, начинается ли строка с определенной подстроки или заканчивается ли строка заданной подстрокой.

Вот как они работают:

```
startswith(prefix[, start[, end]])
```

Этот метод проверяет, начинается ли строка с указанного prefix.

prefix: -это строка, с которой вы хотите выполнить проверку.

start (необязательный): -этот параметр указывает начальный индекс, с которого начать проверку.

end (необязательный): -этот параметр указывает конечный индекс, на котором закончить проверку.

Пример использования startswith():

```
text = "Hello, World!"
result = text.startswith("Hello")
print(result)  # Вывод: True

result = text.startswith("World", 7)  # Начиная с индекса 7
print(result)  # Вывод: True
```

```
endswith(suffix[, start[, end]])
```

этот метод проверяет, заканчивается ли строка указанным suffix.

suffix: -это строка, с которой вы хотите выполнить проверку.

start (необязательный): -этот параметр указывает начальный индекс, с которого начать проверку.

end (необязательный): -этот параметр указывает конечный индекс, на котором закончить проверку.

Пример использования endswith():

```
text = "Hello, World!"
result = text.endswith("World!")
print(result)  # Вывод: True

result = text.endswith("Hello", 0, 5)  # Проверка первых 5 символов
print(result)  # Вывод: True
```

Эти методы возвращают булевое значение (True или False) в зависимости от результата проверки. Они полезны, когда вам нужно выполнить условную операцию на основе начала или конца строки, например, при определении типа файла по его расширению или при проверке префикса URL.

6.find() и index(): в Python используются для поиска подстроки в строке и определения её позиции. Оба метода выполняют похожие задачи, но есть некоторые различия в их поведении.

Вот как они работают:

find(sub[, start[, end]]): -метод find() ищет подстроку sub в строке и возвращает индекс (позицию) первого вхождения этой подстроки. Если подстрока не найдена, метод возвращает -1.

sub: -это подстрока, которую вы ищете в строке.

start (необязательный): -этот параметр указывает начальный индекс, с которого начать поиск.

end (необязательный): -этот параметр указывает конечный индекс, на котором закончить поиск.

Пример использования find():

```
text = "Hello, World!"
index = text.find("World")
print(index)  # Вывод: 7

index = text.find("Python")
print(index)  # Вывод: -1, так как "Python" не найден
```

index(sub[, start[, end]]): -метод index() также ищет подстроку sub в строке и возвращает индекс первого вхождения этой подстроки. Однако, если подстрока не найдена, метод генерирует исключение ValueError.

sub: -это подстрока, которую вы ищете в строке.

start (необязательный): -этот параметр указывает начальный индекс, с которого начать поиск.

end (необязательный): -этот параметр указывает конечный индекс, на котором закончить поиск.

Пример использования index():

```
text = "Hello, World!"
index = text.index("World")
print(index)   # Вывод: 7

try:
    index = text.index("Python")
except ValueError:
    print("Подстрока не найдена")
```

Основное различие между find() и index() заключается в обработке случая, когда подстрока не найдена. find() возвращает -1, в то время как index() генерирует исключение. Поэтому, если вы не уверены, будет ли подстрока найдена, использование find() может быть более безопасным.

7.count(): в Python используется для подсчета количества непересекающихся вхождений подстроки в строке. Этот метод полезен, когда вам нужно определить, сколько раз определенная подстрока встречается в тексте.

Вот синтаксис метода count():

```
string.count(sub[, start[, end]])
```

sub: -это подстрока, которую вы хотите подсчитать в строке.

start (необязательный): -этот параметр указывает начальный индекс, с которого начать подсчет.

end (необязательный): -этот параметр указывает конечный индекс, на котором закончить подсчет.

Пример использования count():

```
text = "Python — потрясающий язык программирования. Python универсален, и Python  интересно изучать."
count_python = text.count("Python")
print(count_python)   # Вывод: 3

substring = "яз"
count_is = text.count(substring, 12, 40)   # Подсчет в определенной части строки
print(count_is)   # Вывод: 1
```

Метод count() возвращает целое число, представляющее количество вхождений подстроки sub в строку. Вы можете использовать этот метод для анализа текстовых данных, подсчета вхождений определенных слов или символов, а также для выполнения подсчетов в определенных частях строки.

Базовые алгоритмы обработки строк:

Базовые алгоритмы обработки строк включают в себя:

1.Поиск подстроки в строке.

2.Замена подстроки в строке.

3.Разделение строки на подстроки.

4.Объединение (конкатенация) строк.

5.Изменение регистра символов.

6.Удаление лишних пробелов и специальных символов.

Эти операции могут быть использованы для решения различных

задач, связанных с обработкой текстовой информации.

С работой со строками в Python вы можете выполнять разнообразные задачи, от обработки текста до ввода/вывода и форматирования данных. Освоение основ работы со строками является ключевым навыком для многих аспектов программирования.

Тема №14. Работа с множествами в Python.

Работа **с множествами (Sets) в** Python представляет собой важную часть работы с данными. Множества представляют собой коллекции уникальных элементов, что означает, что внутри множества не может быть повторяющихся элементов. В этой полной подробной лекции мы рассмотрим основные аспекты работы с множествами в Python, включая создание множеств, основные операции и методы для работы с ними.

Множество (Set) - это структура данных в программировании и математике, которая представляет собой коллекцию уникальных элементов без упорядочения. Основная идея множества заключается в том, что оно содержит только уникальные элементы, и каждый элемент может встречаться в множестве только один раз.

Основные характеристики множества включают:

Уникальность элементов: Множество не может содержать дубликатов. Если вы попытаетесь добавить элемент, который уже существует в множестве, он не будет дублироваться.

Отсутствие упорядочения: Элементы в множестве не имеют определенного порядка. Это означает, что вы не можете получить доступ к элементам множества по индексу, так как индексы не существуют.

Использование фигурных скобок или конструктора: В Python множества можно создавать с помощью фигурных скобок {} или конструктора set(). Например: {1, 2, 3} или set([1, 2, 3]).

Множества полезны для множества задач, таких как удаление дубликатов из списка, проверка наличия элемента, выполнение операций над множествами (объединение, пересечение, разность и симметричная разность) и многое другое. Они представляют собой мощный инструмент для работы с уникальными элементами в данных.

Создание множеств

Множество можно создать, используя фигурные скобки {} или конструктор **set()**.

```python
# Создание множества с использованием фигурных скобок
my_set = {1, 2, 3}

# Создание множества с использованием конструктора set()
another_set = set([4, 5, 6])
```

Обратите внимание, что множество может содержать элементы разных типов данных.

Основные операции с множествами

Добавление элементов в множество

Вы можете добавить элемент в множество, используя метод **add().**

```python
# Создание множества с использованием фигурных скобок
my_set = {1, 2, 3}
# Добовляем элемент в множество
my_set.add(4)

print(my_set)

#ответ
{1, 2, 3, 4}
```

Удаление элементов из множества

Для удаления элемента из множества используют метод **remove().**

```python
# Создание множества с использованием фигурных скобок
my_set = {1, 2, 3}
# Удаляем элемент из множество
my_set.remove(3)

print(my_set)

#ответ
{1, 2}
```

Проверка наличия элемента в множестве

Для проверки, содержится ли элемент в множестве, используйте оператор **in.**

```python
# Создание множества с использованием фигурных скобок
my_set = {1, 2, 3}
# Проверка элемента в множестве
if 2 in my_set:
    print("2 содержится в множестве.")

print(my_set)

#ответ
2 содержится в множестве.
{1, 2, 3}
```

Операции над множествами

Python предоставляет множество операций для работы с множествами, такие как **объединение, пересечение, разность и симметричная разность**. Вот некоторые примеры:

```python
# Объединение множеств
union_set = set1.union(set2)  # или set1 | set2

# Пересечение множеств
intersection_set = set1.intersection(set2)  # или set1 & set2

# Разность множеств
difference_set = set1.difference(set2)  # или set1 - set2

# Симметричная разность множеств
symmetric_difference_set = set1.symmetric_difference(set2)  # или set1 ^ set2
```

Методы для работы с множествами

Python также предоставляет множество полезных методов для работы с множествами. Вот некоторые из них:

```python
my_set = {1, 2, 3}

# Добавление нескольких элементов
my_set.update([4, 5, 6])

# Удаление всех элементов из множества
my_set.clear()

# Копирование множества
new_set = my_set.copy()

# Количество элементов в множестве
count = len(my_set)
```

Итерирование по множеству

Вы можете перебирать элементы множества с помощью **цикла for.**

```python
my_set = {1, 2, 3}

for item in my_set:
    print(item)

#ответ
1
2
3
```

Множества в Python не упорядочены, поэтому порядок элементов может быть произвольным.

Заключение

Множества в Python предоставляют удобный способ работы с уникальными элементами данных. Они полезны во многих сценариях, включая удаление дубликатов, проверку наличия элементов и выполнение операций над множествами. Эта лекция предоставила вам основные знания о работе с множествами в Python.

Тема №15.Работа со списками в Python.

Работа с данными в виде списков является одной из основных задач в Python. **Списки** - это упорядоченные коллекции элементов, которые

могут содержать объекты различных типов данных. Вот некоторые основные аспекты работы со списками в Python:

1.Создание списка:

Для создания списка используется квадратные скобки [], в которых перечисляются элементы списка через запятую. Например:

```
my_list = [1, 2, 3, 4, 5]
```

2.Доступ к элементам списка:

Элементы списка можно получить по их индексу, начиная с 0. Например, чтобы получить первый элемент списка my_list, используйте **my_list[0]**.

3.Изменение элементов списка:

Элементы списка можно изменять, присваивая им новые значения по индексу. Например:

```
my_list = [1, 2, 3, 4, 5]
my_list[0] = 6
print(my_list)
##ответ
[6, 2, 3, 4, 5]
```

4.Длина списка:

Длину списка можно получить с помощью функции **len()**. Например:

```
my_list = [1, 2, 3, 4, 5]
my_list[0] = 6
length = len(my_list)
print(my_list)
print(length)
##ответ
[6, 2, 3, 4, 5]
5
```

5.Добавление элементов:

Чтобы добавить новый элемент в конец списка, используйте метод **append().** Например:

```
my_list = [1, 2, 3, 4, 5]
my_list.append(6)
print(my_list)
##ответ
[1, 2, 3, 4, 5, 6]
```

6.Удаление элементов:

Для удаления элемента из списка используйте метод **remove().** Например:

```
my_list = [1, 2, 3, 4, 5]
my_list.remove(3)
print(my_list)
##ответ
[1, 2, 4, 5]
```

7.Срезы (slicing):

Вы можете получить подсписок списка с помощью срезов. Например, чтобы получить элементы с индексами с 1 по 3 (не включая 3),

используйте **my_list[1:3]**.

8.Итерация по списку:

Для перебора элементов списка можно использовать цикл **for.** Например:

```
my_list = [1, 2, 3, 4, 5]
for item in my_list:
    print(item)
##ответ
1
2
3
4
5
```

9.Проверка наличия элемента:

Чтобы проверить, есть ли определенный элемент в списке, используйте оператор **in**. Например:

```
my_list = [1, 2, 3, 4, 5]
if 5 in my_list:
    print("5 is in the list")

##ответ
5 is in the list
```

10.Сортировка:

Вы можете отсортировать элементы списка с помощью метода **sort().** Например:

```
my_list = [6, 2, 8, 3, 5]
my_list.sort()
print(my_list)
##ответ
[2, 3, 5, 6, 8]
```

11.Копирование списка:

Если вы хотите создать копию списка, используйте срез. Например:

```
my_list = [6, 2, 8, 3, 5]
new_list = my_list[:]

print(new_list)
##ответ
[6, 2, 8, 3, 5]
```

Это основы работы с списками в Python. Списки предоставляют множество возможностей для хранения и обработки данных, и они являются важным элементом в большинстве программ на Python.

Пример:

Проверка наличия элемента в списке:

```python
# Создаем список дней недели
weekdays = ["понедельник", "вторник", "среда", "четверг", "пятница"]

# Проверяем, есть ли "вторник" в списке
if "вторник" in weekdays:
    print("Вторник есть в списке.")
else:
    print("Вторник отсутствует в списке.")


##ответ
Вторник есть в списке.
```

Тема №16. Работа с кортежами в Python.

Что такое кортеж (tuple)?

Кортеж (tuple) - это упорядоченная и неизменяемая структура данных в Python. Каждый элемент кортежа может содержать данные разных типов, и элементы кортежа разделяются запятыми. Кортежи очень похожи на списки **(list)**, но их ключевое отличие заключается в том, что кортежи не могут быть изменены после создания. Это означает, что вы можете использовать кортеж для хранения данных, которые не должны меняться.

1.Создание кортежа:

```python
my_tuple = (1, 2, 3, "Hello", 3.14)
```

2.Доступ к элементам кортежа:

Вы можете получить доступ к элементам кортежа, используя индексацию, как и в списках:

```python
my_tuple = (1, 2, 3, "Hello", 3.14)
element = my_tuple[3]  # Получить четвертый элемент кортежа
print(element)
##ответ
Hello
```

3.Длина кортежа:

Чтобы узнать длину кортежа, вы можете использовать функцию **len():**

```
my_tuple = (1, 2, 3, "Hello", 3.14)
length = len(my_tuple)  # Получить длину кортежа
print(length)
##Ответ
5
```

6.Итерация по кортежу:

Вы можете использовать цикл for для итерации по элементам кортежа:

```
my_tuple = (1, 2, 3, "Hello", 3.14)
for item in my_tuple:
    print(item)
##Ответ
1
2
3
Hello
3.14
```

7.Проверка наличия элемента в кортеже:

Чтобы проверить, содержит ли кортеж определенный элемент, вы можете использовать оператор in:

```
my_tuple = (1, 2, 3, "Hello", 3.14)
if 3 in my_tuple:
    print("3 is in the tuple")

##Ответ
3 is in the tuple
```

8.Срезы (slicing) кортежей:

Вы можете использовать срезы для получения подмножества элементов кортежа:

```
my_tuple = (1, 2, 3, "Hello", 3.14)
subset = my_tuple[1:3]  # Получить подмножество с элементами с индексами 1 и 2
print(subset)
##Ответ
(2, 3)
```

9.Неизменяемость кортежей:

Как было сказано ранее, кортежи неизменяемы. Это означает, что после создания кортежа, вы не можете изменить его элементы. Если вы попытаетесь сделать это, вы получите ошибку.

10.Методы кортежей:

Кортежи имеют только два метода: **count()** и **index()**. Метод **count()** используется для подсчета количества вхождений элемента в кортеж, а метод **index()** используется для поиска индекса первого вхождения элемента.

Примеры:

```
my_tuple = (1, 2, 2, 3, 4, 2,2)
count = my_tuple.count(2)  # count равен 4
index = my_tuple.index(3)  # index равен 3
print(count)
print(index)
#Ответ
4
3
```

Это основы работы с кортежами в Python. Кортежи полезны, когда вам нужно создать не изменяемую последовательность данных.

Пример который демонстрирует работу с кортежами для хранения данных о книгах в библиотеке:

```
# Создание кортежей с информацией о книгах
book1 = ("Таң алдында", "Аалы Токомбаев", 1962)
book2 = ("Акылман бала", "Касымаалы Жантошов", 1995)
book3 = ("Мастер и Маргарита", "Михаил Булгаков", 1967)
book4 = ("Ак кеме", "Чынгыз Айтматов", 1949)

# Создание кортежа, содержащего информацию о книгах
library = (book1, book2, book3, book4)

# Итерация по библиотеке и вывод информации о книгах
for book in library:
    title, author, year = book
    print(f"Название: {title}")
    print(f"Автор: {author}")
    print(f"Год издания: {year}")
    print()

# Поиск книги по автору
author_name = "Чынгыз Айтматов"
author_books = []
for book in library:
    title, author, year = book
    if author == author_name:
        author_books.append(title)

if author_books:
    print(f"Книги автора {author_name}: {', '.join(author_books)}")
else:
    print(f"Книги автора {author_name} не найдены в библиотеке.")
```

 # Создание кортежей с информацией о книгах: в этой части кода мы создаем четыре кортежа (book1, book2, book3, book4), каждый из которых содержит информацию о книге: название, автора и год издания. Затем мы создаем кортеж library, который содержит эти кортежи книг.

 # Итерация по библиотеке и вывод информации о книгах: Здесь мы используем цикл for, чтобы перебрать все книги в library. В каждой итерации мы распаковываем кортеж book в отдельные переменные title, author и year, и затем выводим информацию о каждой книге, включая название, автора и год издания.

 # Поиск книги по автору: здесь мы выполняем поиск книг по имени автора (в данном случае, автора "Джордж Мартин"). Мы создаем пустой список author_books, затем используем цикл for, чтобы проверить каждую книгу в библиотеке. Если автор книги совпадает с author_name, то мы добавляем название этой книги в список author_books. После завершения цикла, мы проверяем, были ли найдены книги автора, и выводим соответствующее сообщение.

Тема №17. Работа со словарями в Python

Предоставляем краткую лекцию о работе со словарями (dictionaries) в Python. **Словари** - это одна из важных структур данных в Python, они представляют собой неупорядоченные коллекции пар "ключ-значение". Вот некоторые основные понятия и операции, связанные с работой со словарями:

1.Создание словаря:

Для создания словаря используется фигурные скобки {} или конструктор **dict()**. Каждая пара "ключ-значение" разделяется двоеточием, а элементы разделяются запятой. Например:

```
my_dict = {'имя': 'Алиса', 'возраст': 25, 'город': 'Токмок'}
```

2.Доступ к элементам словаря:

Для получения значения по ключу используйте квадратные скобки **[]** или метод **get()**. Например:

```
my_dict = {'имя': 'Алиса', 'возраст': 25, 'город': 'Токмок'}
name = my_dict['имя']
age = my_dict.get('возраст')
print(name)
print(age)
```

Если ключа нет в словаре, использование **get()** вернет None, а использование квадратных скобок вызовет ошибку.

3.Изменение и добавление элементов:

Вы можете изменить значение элемента в словаре, просто указав ключ и присваивая новое значение:

```
my_dict = {'имя': 'Алиса', 'возраст': 25, 'город': 'Токмок'}
my_dict['возраст'] = 26  # Изменение возраста
print(my_dict)
# ответ
{'имя': 'Алиса', 'возраст': 26, 'город': 'Токмок'}
```

Чтобы добавить новый элемент, просто укажите новый ключ и его значение:

```
my_dict = {'имя': 'Алиса', 'возраст': 25, 'город': 'Токмок'}
my_dict['возраст'] = 26  # Изменение возраста
my_dict['пол'] = 'женский'  # Добавление нового элемента
print(my_dict)
# ответ
{'имя': 'Алиса', 'возраст': 26, 'город': 'Токмок', 'пол': 'женский'}
```

4.Удаление элементов:

Для удаления элемента из словаря используйте оператор **del**:

```
my_dict = {'имя': 'Алиса', 'возраст': 25, 'город': 'Токмок'}
del my_dict['город']  # Удаление элемента по ключу

print(my_dict)
# ответ
{'имя': 'Алиса', 'возраст': 25}
```

5.Проверка наличия ключа:

Чтобы проверить, есть ли ключ в словаре, используйте оператор **in:**

```
my_dict = {'имя': 'Алиса', 'возраст': 25, 'город': 'Токмок'}
if 'город' in my_dict:
    print("Город есть в словаре")
print(my_dict)
# ответ
Город есть в словаре
{'имя': 'Алиса', 'возраст': 25, 'город': 'Токмок'}
```

6.Получение ключей и значений:

Вы можете получить список всех ключей и всех значений из словаря с помощью методов **keys()** и **values()** соответственно:

```
my_dict = {'имя': 'Алиса', 'возраст': 25, 'город': 'Токмок'}
keys = my_dict.keys()
values = my_dict.values()
print(keys)
print(values)
# ответ
dict_keys(['имя', 'возраст', 'город'])
dict_values(['Алиса', 25, 'Токмок'])
```

7.Итерация по словарю:

Вы можете перебирать все пары "ключ-значение" с помощью цикла **for:**

```
my_dict = {'имя': 'Алиса', 'возраст': 25, 'город': 'Токмок'}
for key, value in my_dict.items():
    print(f"{key}: {value}")

# ответ
имя: Алиса
возраст: 25
город: Токмок
```

Это основы работы со словарями в Python. Словари очень удобны и эффективны для хранения и манипуляции данными. Вы можете использовать их для хранения информации, конфигураций, кэширования и многих других задач.

Тема №18.Обработка вложенных последовательностей (ВП). Базовые алгоритмы ВП.

Формирование вложенных последовательностей

Ранее мы познакомились с обработкой таких последовательностей данных с помощью языка программирования Python, как списки и кортежи. Однако очень часто информация находится в таблицах, данные в которых представлены в виде последовательности строк. Получается, что строки как бы вложены друг в друга, т. е. одна последовательность данных вложена в другую.

Можно провести аналогию с другими языками программирования, когда обработка информации, размещенной в таблице, происходит с помощью инициализации так называемых двумерных массивов (матриц).

На рис. 1 представлен общий вид квадратной матрицы 4x4. Каждый элемент матрицы имеет имя А. Первый индекс - это номер строки, второй индекс-номер столбца. Диагональ матрицы A_{00}-A_{44} называется главной, а диагональ A_{04} - A_{40} - побочной.

00	“01	“02	“03	“04
10	a_{11}	a_{12}	a_{13}	a_{14}
20	a_{21}	a_{22}	a_{23}	a_{24}
30	a_{31}	a_{32}	a_{33}	a_{34}
40	a_{41}	a_{42}	a_{43}	a_{44}

Рис 1. Общий вид квадратной матрицы

Возможности языка Python по обработке вложенных последовательностей удивят человека, привыкшего к классической обработке матриц в таких языках программирования, как C#, Microsoft Visual Basic, Delphi и др. Например, вложенная последовательность в Python может содержать одновременно список и кортеж, строковые и числовые данные и т. д.

Следует отметить, что точно так же, как и при обработке матриц, операции по обработке, вводу-выводу элементов вложенных последовательностей в Python осуществляются с использованием сложного циклического процесса.

В нижеприведенном примере (код программы 1) список а состоит из трех элементов:

первый - список 1, 9, 6, 2,
второй - список 5, 6, 7, 12
и третий - список 7, 8, 9, 28.
a=[[1,9,6,2],[5,6,7,12],[7,8,9,28]]

Выводить на экран такой список будем, организовав первый цикл по номеру строки, а второй цикл - по элементам внутри строки.

Код программы 1:

```
a=[[1,9,6,2],[5,6,7,12],[7,8,9,28]]
for i in range(len(a)):
    for j in range(len(a[i])):
        print(a[i][j], end=" ")
    print()
```

Результат 1:

```
1 9 6 2
5 6 7 12
7 8 9 28
```

```
print(a[0])
```

```
[1, 9, 6, 2]
```

Обратиться к элементу вложенного списка можно, указав его индекс. Например,
На экран будет выведен список:

Чтобы извлечь третий элемент первого списка, следует указать два индекса: первый - номер списка, в котором находится нужный элемент; второй - позиция этого элемента в списке. Так, для того чтобы извлечь третий элемент списка (цифра 2) в первом списке, напишем такой оператор:

результат :
```
print(a[0][3])
```

```
2
```

Для создания вложенных последовательностей можно использовать вложенные генераторы, разместив генератор списка, являющегося строкой, внутри генератора строк (код программы 2):

```
n = 5
m = 5
a = [[i*j for j in range(m)] for i in range (n)]
for i in range(n):
    for j in range(m):
        print(a[i][j], end=" ")
    print()
```

Результат 2:

```
0 0 0 0 0
0 1 2 3 4
0 2 4 6 8
0 3 6 9 12
0 4 8 12 16
```

Немного видоизменив предыдущий код за счет применения функции sample, будем из исходной последовательности элементов списка возвращать указанное количество элементов (код программы 3). Лучше, если это будет число пять, потому что матрица 5x5, как правило, используется в учебных целях при решении задач. При этом воспользуемся модулем random, подключив его с помощью инструкции import.Код программы 3:

```
import random
sum = 0
n = 5
m = 5
a = [[i+j for j in  random.sample(range(100),5)] for i in random.sample(range(100),5)]
for i in range(n):
    for j in range(m):
        print(a[i][j], end=" ")
    print()
```

Результат 3:

```
86 126 87 145 166
42 99 72 109 96
96 94 61 48 84
140 133 79 111 130
158 122 145 171 161
```

Задача 1. Во вложенной последовательности заранее заданных целых чисел найдите сумму всех элементов. Разработка алгоритма решения задачи представлена на рис. 2.

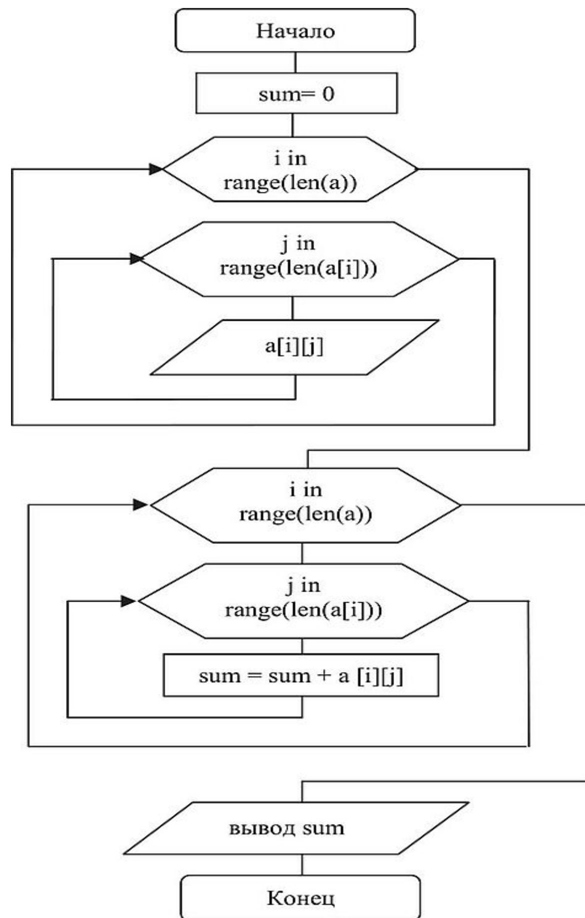

Рис.2. Алгоритм решения задачи.

Ниже приведен код программы, отвечающий за решение задачи, и результат ее выполнения.Код программы 4:

```
sum = 0
a=[[1,9,6,2],[5,6,7,12],[7,8,9,28]]
for i in range(len(a)):
    for j in range(len(a[i])):
        print(a[i][j], end=" ")
    print()
for i in range(len(a)):
    for j in range(len(a[i])):
        sum = sum + a[i][j]
print('сумма элементов списка -',sum)
```

Результат 4:

```
1 9 6 2
5 6 7 12
7 8 9 28
сумма элементов списка - 100
```

В код программе 5 решение задачи, приводящее к нахождению суммы элементов вложенной последовательности, организовано не по индексу элемента, а по значениям списка, где row - строка, elem - значение элемента в строке.

Код программы 5:

```
sum = 0
a=[[1,9,6,2],[5,6,7,12],[7,8,9,28]]
for i in a:
    for j in i:
        print(j, end=" ")
    print()
for row in a:
    for elem in row:
        sum = sum + elem
print('сумма элементов списка -',sum)
```

Результат 5:

```
1 9 6 2
5 6 7 12
7 8 9 28
сумма элементов списка - 100
```

Базовые алгоритмы обработки вложенных последовательностей

В силу принятого нами подхода работы со списками можно выделить ряд приемов, позволяющих реализовать основные задачи обработки вложенных последовательностей, а именно:

1. Нахождение количества элементов вложенной последовательности при заданном условии;
2. Нахождение суммы значений элементов вложенной последовательности при заданном условии;
3. Нахождение произведения значений элементов вложенной последовательности при заданном условии;
4. Поиск экстремальных значений элементов вложенной последовательности (поиск максимального и/или минимального значения);
5. Обмен столбцов элементов вложенной последовательности;
6. Обмен строк элементов вложенной последовательности;
7. Удаление заданной строки вложенной последовательности;
8. Замена значений элементов вложенной последовательности.

Ниже описывается реализация перечисленных алгоритмов обработки вложенных последовательностей. Первые четыре способа чрезвычайно просты, данные алгоритмы неоднократно рассматривались при решении задач в предыдущих разделах и в настоящее время не требуют комментариев. Однако стоит отметить, что в первых четырех кодах программы очередной элемент вложенной последовательности сравнивается с числом 10, однако на практике лучше организовать запрос данных от пользователя.

1.Нахождение количества элементов вложенной последовательности при заданном условии:

```python
# Нахождение количества элементов вложенной последовательности
import random
kol = 0
n = 5
m = 5
a = [[i+j for j in random.sample(range(15),5)] for i in random.sample(range(15),5)]
for i in range(n):
    for j in range(m):
            print(a[i][j], end = " ")
    print()
for i in range(n):
    for j in range(m):
          if a[i][j]>=15:
              kol=kol+1
print("Количество элементов вложенной последовательности = ", kol)
```

Результат:

```
5 3 13 10 8
16 8 14 18 10
12 15 11 19 18
20 16 13 21 24
10 12 13 8 4
Количество элементов вложенной последовательности =  25
```

2.Нахождение суммы значений элементов вложенной последовательности при заданном условии:

```python
# Нахождение суммы значений элементов вложенной последовательности
import random
sum = 0
n = 5
m = 5
a = [[i+j for j in random.sample(range(15),5)] for i in random.sample(range(15),5)]
for i in range(n):
    for j in range(m):
            print(a[i][j], end = " ")
    print()
for i in range(n):
    for j in range(m):
          if a[i][j]>=15:
              sum=sum+a[i][j]
print("Сумма значений элементов вложенной последоваттельности = ", sum)
```

Результат:

```
7 13 6 17 16
14 16 18 7 19
16 15 23 26 18
7 6 12 8 13
2 14 5 15 9
Сумма значений элементов вложенной последоваттельности =  322
```

3.Нахождение произведения значений элементов вложенной последовательности при заданном условии:

```python
# Нахождение произведения значений элементов вложенной последовательности
import random
pr = 1
n = 5
m = 5
a = [[i+j for j in random.sample(range(15),5)] for i in random.sample(range(15),5)]
for i in range(n):
    for j in range(m):
            print(a[i][j], end = " ")
    print()
for i in range(n):
    for j in range(m):
          if a[i][j]>=15:
              pr = pr*a[i][j]
print("Произведения значений элементов вложенной последоваттельности = ", pr)
```

Результат:
```
10 7 8 16 12
5 8 12 10 3
10 5 7 9 16
14 27 25 26 24
20 11 14 17 10
Произведения значений элементов вложенной последоваттельности =  36661248000
```

4.Поиск экстремальных значений элементов вложенной последовательности (поиск максимального и/или минимального значения):

```
# Поиск экстремальных значений элементов вложенной последовательности
import random
max = -32768
min = 32767
n = 5
m = 5
a = [[i+j for j in random.sample(range(15),5)] for i in random.sample(range(15),5)]
for i in range(n):
    for j in range(m):
            print(a[i][j], end = " ")
    print()
for i in range(n):
    for j in range(m):
          if a[i][j]>=max:
              max = a[i][j]
          if a[i][j]<min:
              min = a[i][j]
print("Максимальный элемент вложенной последоваттельности = ", max)
print("Минимальный элемент вложенной последоваттельности = ", min)
```

Результат:
```
15 11 16 10 23
6 14 15 17 16
8 10 15 12 3
15 14 11 2 8
18 16 22 21 17
Максимальный элемент вложенной последоваттельности =  23
Минимальный элемент вложенной последоваттельности =  2
```

5. Обмен столбцов элементов вложенной последовательности

Задача 2. Во вложенной последовательности произвольных чисел поменяйте местами первый и четвертый столбцы. Выведите обе вложенные последовательности на экран.

Комментарий. После генерации элементов вложенной последовательности следует организовать цикл и выполнить в нем три оператора, отвечающих за перестановку столбцов вложенной последовательности:

k = a [i] [1]   a[i][l] = a[i][4]   a[i][4] = к

Поскольку при первом вхождении в цикл параметр цикла i примет значение, равное нулю, во вспомогательную ячейку к отправляется нулевой элемент первого столбца, т. е. элемент, имеющий координаты a_{0i}. На его место приходит элемент с координатами a_{04}. На освободившееся место перейдет элемент, находящийся в ячейке к. Произошел обмен. Так как все действия осуществляются в цикле for i in range(n), происходит обмен элементов первого и четвертого столбцов. Разработка алгоритма решения задачи представлена на рис.2.

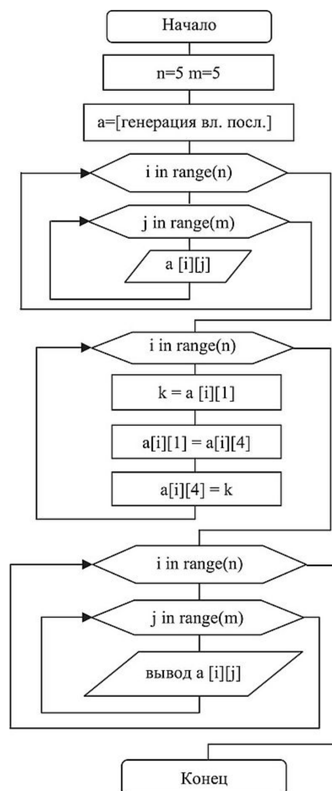

Рис.2. Алгоритм решения задачи

Код программы, отвечающий за решение задачи:

```
import random
n = 5
m = 5
a = [[i+j for j in random.sample(range(15),5)] for i in random.sample(range(15),5)]
for i in range(n):
    for j in range(m):
        print(a[i][j], end = " ")
    print()
for i in range(n):
    k=a[i][1]
    a[i][1]=a[i][4]
    a[i][4]=k
    print()
for i in range(n):
    for j in range(m):
        print(a[i][j], end= " ")
    print()
```

Результат:

```
9 1 5 3 14
20 18 22 10 17
4 5 1 11 9
4 12 8 7 5
21 12 20 14 22



9 14 5 3 1
20 17 22 10 18
4 9 1 11 5
4 5 8 7 12
21 22 20 14 12
```

Первый столбец исходной вложенной последовательности поменяли с четвертым столбцом.

6.Обмен строк во вложенной последовательности

В данной задаче происходит обмен тех строк, номера которых (2 и 4) задает пользователь. Алгоритмически метод ничем не отличается от рассмотренного выше. Ниже приведен код программы, отвечающий за решение задачи:

```
n = 5
m = 5
a = [[i+j for j in random.sample(range(15),5)] for i in random.sample(range(15),5)]
for i in range(n):
    for j in range(m):
        print(a[i][j], end = " ")
    print()
n_st1=int(input("Введите номер строки, которая подлежит замене "))
n_st2=int(input("Введите номер строки, которую обеняем"))
for j in range(n):
    bufer=a[n_st2][j]
    a[n_st2][j]=a[n_st1][j]
    a[n_st1][j]=bufer
for i in range(n):
    for j in range(m):
        print(a[i][j], end= " ")
    print()
```

Результат:

```
2 6 1 12 10
10 11 12 19 8
15 19 16 18 11
13 18 20 8 9
12 19 14 8 17
Введите номер строки, которая подлежит замене 2
Введите номер строки, которую обеняем4
2 6 1 12 10
10 11 12 19 8
12 19 14 8 17
13 18 20 8 9
15 19 16 18 11
```

Результат работы программы: вторую строку исходной вложенной последовательности поменяли с четвертой строкой.

7.Удаление заданной строки и столбца во вложенной последовательности

Удаление заданной строки и столбца происходит с помощью команды del. Задачу удаления столбца можно решить аналогично. Ниже приведен код программы, отвечающий за решение задачи:

```
#Удаление заданной строки и столбца во вложенной последовательности
import random
n = 5
m = 5
a = [[i+j for j in random.sample(range(15),5)] for i in random.sample(range(15),5)]
for i in range(n):
    for j in range(m):
        print(a[i][j], end = " ")
    print()
n_st=int(input("Введите номер строки и столбца , которую хотите удалить "))
for i in range(len(a)):
        del a[i][n_st]      # удаляет столбец
del a[n_st]         # удаляет строку
for i in range(len(a)):
    for j in range(len(a[i])):
        print(a[i][j], end= " ")
    print()
```

Результат:

```
6 9 17 5 4
18 19 13 16 24
9 18 4 8 11
12 9 11 8 10
19 24 17 15 22
Введите номер строки и столбца , которую хотите удалить 4
6 9 17 5
18 19 13 16
9 18 4 8
12 9 11 8
```

Результат работы программы: четвертую строку и четвертый столбец исходной вложенной последовательности удалили.

8. Замена значений элементов вложенной последовательности.

Задача. Все элементы исходной вложенной последовательности замените нулем, а каждый элемент главной диагонали замените его номером.

Вполне логично первоначально осуществить проверку условия: «Находится ли элемент вложенной последовательности на главной диагонали?», т. е. if i==j, а затем выполнить прямое присваивание

a[i||j|=i (где i - номер элемента на главной диагонали) или оператор a[i][j]=0, который заменит элементы исходной вложенной последовательности нулем. Ниже приведен код программы, отвечающий за решение задачи:

```
#Замена значений элементов вложенной последовательности.
import random
n = 5
m = 5
a = [[i+j for j in random.sample(range(15),5)] for i in random.sample(range(15),5)]
for i in range(n):
    for j in range(m):
        print(a[i][j], end = " ")
    print()

for i in range(n):
    for j in range(n):
        if i==j: # проверка находится ли элемент на главной диоганали?
            a[i][j]=i
        else:
            a[i][j]=0 # Замена остальных элементов на нуль
print()

for i in range(n):
    for j in range(m):
        print(a[i][j], end = " ")
    print()
```

Результат:

```
17 9 15 22 10
11 19 22 13 20
18 24 15 14 26
15 10 14 18 6
19 9 14 15 6

0 0 0 0 0
0 1 0 0 0
0 0 2 0 0
0 0 0 3 0
0 0 0 0 4
```

Результат работы программы: все элементы вложенной последовательности заменены нулями, элементы главной диагонали - порядковыми номерами.

Методическая разработка аудиторных форм работы (Краткое содержание практических занятий)

Практическая работа №7. Работа со строками в Python.Методы работы со строками

Цель работы: познакомится с методами работы со строками.

Учащийся должен:

Владеть: Навыками составления линейных алгоритмов на языке программирования Python с использованием строковых данных;

Уметь: Применять функции и методы строк при обработке строковых данных;

Знать: Операции и методы обработки строк.

Вопросы для защиты работы

1.Как в программах используются переменные строкового типа?

2.Как преобразовать число в строку и обратно?

3.Какой индекс имеет первый символ строки?

4.Как удалить лишние пробелы в строке?

5.Какие методы используются для обработки строк?

Общие теоретические сведения

Строка — базовый тип представляющий из себя неизменяемую последовательность символов; str от «string» — «строка».

Следует отметить, что первоначальные навыки работы со строками в языке программирования Python у нас уже сформированы. Мы знаем, как сделать приглашение к вводу данных или вывести ответ в программе. В этой главе мы познакомимся с функциями и методами, предназначенными для работы со строками.

Строка в Python - это объект класса str, поэтому ранее, для того чтобы работать с числовыми данными, мы применяли функции приведения типов, например, int.

Строки индексируются с нуля, т. е., если имеется оператор stroka="python", то первый символ в строке нулевой и обращение к нему будут записываться как stroka[0]. В то же время к элементам строки в Python может обращаться, указывая отрицательные индексы, например, оператор print(stroka[-6]) есть ничто иное, как вывод символа p на экран.

Код программы:

```
stroka = "python"
print('0 элемент = ',stroka[0])
```

Резудьтат:

```
0 элемент =  p
```

Код программы:

```
stroka = "python"
print('-6 элемент = ',stroka[-6])
```

Результат:

```
-6 элемент =  p
```

При обработке строки к ней можно обратиться непосредственно в цикле, как это показано в следующем коде программы:

```
stroka = "python"
for i in stroka:
    print(i, end=" ")
print('\n 0 элемент = ',stroka[0])
```

```
p y t h o n
 0 элемент =  p
```

Функции и методы работы со строками

Функция или метод	**Назначение**
S1 + S2	**Конкатенация (сложение строк)**
S1 * 3	**Повторение строки**

S[i]	Обращение по индексу
S[i:j:step]	Извлечение среза
len(S)	Длина строки
S.join(список)	Соединение строк из последовательности str через разделитель, заданный строкой
S1.count(S[, i, j])	Количество вхождений подстроки s в строку s1. Результатом является число. Можно указать позицию начала поиска i и окончания поиска *j*
S.find(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.index(str, [start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает ValueError
S.rindex(str, [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает ValueError
S.replace(шаблон, замена)	Замена шаблона
S.split(символ)	Разбиение строки по разделителю
S.upper()	Преобразование строки к верхнему регистру
S.lower()	Преобразование строки к нижнему регистру

Рассмотрим основные методы работы со строками.

1. **Метод upper().**

Синтаксис метода: stroka.upper(). Преобразует все символы строки к верхнему регистру. Например:

```
stroka="python"
stroka1=stroka.upper()
print(stroka1)
```

Результат:

```
PYTHON
```

2. Метод lower().

Синтаксис метода: stroka.lower(). Преобразует все символы строки к нижнему регистру. Например:

```
stroka="PYTHON"
stroka1=stroka.lower()
print(stroka1)
```

Результат:

```
python
```

3. Метод swapcase().

Синтаксис метода: stroka.swapcase(). Преобразует все символы строки, записанные в нижнем регистре - в верхний и наоборот.

Например:

```
stroka="PytHoN"
stroka1=stroka.swapcase()
print(stroka1)
```

Результат:

```
pYThOn
```

4. Метод capitalize().

Синтаксис метода: stroka.capitalize(). Преобразует первую букву в строке в верхний регистр,а все остальные в нижний.

Например:

```
stroka="pytHoN"
stroka1=stroka.capitalize()
print(stroka1)
```

Результат:

```
Python
```

5. Метод title().

Синтаксис метода: stroka.title(). Преобразует все первые буквы в строке в верхний регистр .

Например:

```
stroka="python"
stroka1=stroka.capitalize()
print(stroka1)
```

Результат:

```
Python
```

6. Метод startswith().

Синтаксис метода: stroka.startswith(podstroka). Проверяет, начинается ли строка stroka с указанной подстроки podstroka. Например, в данном примере проверяется условие наличия подстроки "Язы" в начале строки "Язык программирования Python".

```
stroka= 'Язык программирования Python'
podstroka='thon'
n=stroka.startswith(podstroka)
print(n)
```

Результат:

```
False
```

```
stroka= 'Язык программирования Python'
podstroka='Язы'
n=stroka.startswith(podstroka)
print(n)
```

Результат:

```
True
```

Результатом данного кода будет значение True.

7. Метод endswith().

Синтаксис метода: stroka.endswith (podstroka). Проверяет, заканчивается ли строка stroka указанной подстрокой podstroka. Например, в данном примере проверяется условие наличия подстроки "thon" в начале строки "Язык программирования Python".

```
stroka= 'Язык программирования Python'
podstroka='thon'
n=stroka.endswith(podstroka)
print(n)
```

Результат:

```
True
```

8. Метод replace().

Синтаксис метода: stroka.replace(old, new), где old - подстрока для замены, new - новая подстрока. Метод находит и заменяет в строке stroka подстроку old подстрокой new. Например, в данном примере оператор stroka = stroka.Replace ("еще","вот") возвратит строку "Эх раз, вот раз, вот много, много раз".

```
stroka= 'Эх раз, еще раз, еще много,много раз'
stroka=stroka.replace('еще','вот')
print(stroka)
```

Результат:

```
Эх раз, вот раз, вот много,много раз
```

9. Метод rfind().

Синтаксис метода: stroka.rfind(podstr), где podstr - подстрока. Метод возвращает позицию последнего вхождения подстроки в строку. Если подстрока не обнаружена, то возвращается значение -1. После параметра podstr могут быть указаны начальная позиция и конечная позиции в строке, где следует искать подстроку. Эти параметры необязательны, и если отсутствует начальная позиция, то поиск будет осуществлен с начала строки. Так, в нижеприведенном примере, последнее вхождение подстроки "еще" будет зафиксировано на позиции 17. Строка, напомним, индексируется с нуля, пробелы и знаки препинания являются символами.

```
stroka= 'Эх раз, еще раз, еще много,много раз'
stroka=stroka.rfind('еще')
print(stroka)
```

Результат:

```
17
```

10. Метод find().

Синтаксис метода: stroka.flnd(podstr), где podstr - подстрока. В отличие от предыдущего метода, метод find() возвращает номер позиции, с которого начинается вхождение подстроки в строку. Если подстрока не обнаружена, то возвращается значение -1. Соответственно, если изменить предыдущую программу и вместо метода rfind() использовать метод find(), то первое вхождение подстроки "еще" в строку "Эх раз, еще раз, еще много, много раз" будет начинаться на позиции 8.

```
stroka= 'Эх раз, еще раз, еще много,много раз'
stroka=stroka.find('еще')
print(stroka)
```

Результат:

```
8
```

11. Метод count().

Синтаксис метода: stroka.count(podstr), где podstr - подстрока. Метод возвращает количество вхождений подстроки podstr в строку stroka. Таким образом, в нижеследующем примере ответом, выведенным на экран оператором print, будет число 2, поскольку подстрока "еще" входит два раза в строку "Эх раз, еще раз, еще много, много раз".

```
stroka= 'Эх раз, еще раз, еще много,много раз'
stroka=stroka.count('еще')
print(stroka)
```

Результат:

```
2
```

Синтаксис метода: stroka.strip(). Метод удаляет начальные и конечные пробелы в строке stroka. В нижеприведенном примере в исходной строке присутствуют пробелы в начале и в конце строки. Ниже представлен результат выполнения программы, где пробелы с помощью метода strip() удалены из исходной строки.

```
stroka= '   Эх раз, еще раз, еще много,много раз   '
print(stroka)
stroka=stroka.strip()
print(stroka)
```

Результат:

```
   Эх раз, еще раз, еще много,много раз
Эх раз, еще раз, еще много,много раз
```

13.Методы lstrip() и rstrip().

Их синтаксис и действие аналогичны рассмотренному методу strip(), с той разницей, что метод lstrip() удаляет символы пробела слева от начала исходной строки, а метод rstrip() - в конце исходной строки.

Отметим, что с помощью методов strip(), rstrip() и lstrip() можно удалять не только пробелы. Так, если в качестве параметра одного из методов указать символ или последовательность символов, то произойдет его (их) удаление. Например, в нижеследующем примере мы указали в качестве параметра в методе strip() символы "Эх". Соответственно, метод возвратит исходную строку, но уже без указанных символов.

```
stroka= 'Эх раз, еще раз, еще много,много раз'
stroka=stroka.strip('Эх')
print(stroka)
```

Результат:

```
 раз, еще раз, еще много,много раз
```

14. Метод split().

Синтаксис метода: stroka.split(). Метод разделяет строку на подстроки и добавляет их в список. Если в качестве параметра присутствует разделитель, то строка будет разбита на подстроки в соответствии с указанным разделителем. В случае его отсутствия разделителем считается пробел. В ниже следующем коде, исходная строка методом split() преобразована в список, разделителем служит пробел. Далее уже над списком

выполняется списочный метод remove(), который удаляет тот элемент списка, который вводит пользователь с клавиатуры.

```
stroka= 'Эх раз, еще раз, еще много, много раз'
spisok=stroka.split()
print(spisok)
element=input('введите элемент списка, который хотите удалить')
for i in spisok:
    if element in spisok:
        spisok.remove(element)
print(spisok)
```

Результат:

```
['Эх', 'раз,', 'еще', 'раз,', 'еще', 'много,', 'много', 'раз']
введите элемент списка, который хотите удалитьраз
['Эх', 'раз,', 'еще', 'раз,', 'еще', 'много,', 'много']
```

15. Метод list().

Синтаксис метода: разделитель(stroka), где list - список или кортеж. Еще одним способом преобразования строки в список является использование функции list, а обратное преобразование осуществляется методом join(). Такие преобразования необходимы для того, чтобы изменить элемент строки по индексу, поскольку изначально строки неизменяемы.

В нижеприведенном примере из исходной строки с помощью функции list() происходит формирование списка. Попробуем заменить элемент исходного списка на введенный с клавиатуры и для этого осуществляем соответствующие запросы с помощью функции input(). Вычисляем длину списка с помощью функции len() и организуем просмотр списка с помощью оператора цикла for. Если очередной элемент списка будет равен введенному элементу, то произойдет замена исходного элемента списка element на элемент elementl. Затем, используя метод join(), преобразуем список в строку и выводим ее на экран .

```
stroka= 'Эх раз, еще раз, еще много, много раз'
spisok=list(stroka)
print(spisok)
element=input('введите элемент списка, который хотите заменить')
element1=input('введите элемент списка, на который хотите заменить')
n=len(spisok)
for i in range (0,n):
    if  spisok[i]==element:
        spisok[i]=element1
        strokal="".join(spisok)
print(strokal)
```

Результат:

```
['Э', 'х', ' ', 'р', 'а', 'з', ',', ' ', 'е', 'щ', 'е', ' ', 'р', 'а', 'з', ',', ' ', 'е', 'щ', 'е', ' ', 'м', 'н', 'о', 'г', 'о', ',', ' ', 'м', 'н', 'о', 'г', 'о', ' ', 'р', 'а', 'з']
введите элемент списка, который хотите заменитьа
введите элемент списка, на который хотите заменитья
Эх ряз, еще ряз, еще много, много ряз
```

Задания для самостоятельной работы:

Вариант 1

Определение количества символов

Задача. Пользователь вводит исходную строку stroka и символ simvol. Подсчитайте, сколько раз заданный символ встречается в исходной строке.

Вариант 2

Замена символов в строке

Задача. Пользователь вводит исходную строку row и символ simv. Замените пробелы в исходной строке указанным символом.

Вариант 3

Удаление символов в строке

Задача. Пользователь вводит исходную строку row и символ simv. Удалите в исходной строке указанный пользователем символ.

Вариант 4

Вставка символа в строку

Задача. Пользователь вводит исходную строку row. В исходную строку необходимо добавить символ simvoll после символа simvol, который пользователь вводит с клавиатуры.

Вариант 5

Анализ символа на принадлежность к группе

Задача. Пользователь вводит исходную строку row. Определите, сколько в ней символов simv_l и simv_2, которые пользователь вводит с клавиатуры.

Вариант 6

Обращение строки

Задача. Пользователь вводит строку stroka. Переверните ее (т. е. запишите наоборот).

Вариант 7

Алфавитная выборка

Задача. Имеется заранее заданный алфавит. Пользователь вводит строку stroka. Выберите из строки символы, используемые в алфавите, и выведите их на экран.

Вариант 8

Срезы строк

Задача. Из исходной строки получите символы, расположенные между заданными начальным и конечным значениями.

Вариант 9

Задача . Пользователь вводит строку row, содержащую несколько слов, между которыми один или несколько пробелов. Требуется найти количество символов в самом длинном слове.

Вариант 10

Задача . Пользователь вводит строку row, содержащую несколько слов, разделенных запятой, но пробел после запятой отсутствует. Результатом работы программы должна стать исправленная строка.

Практическая работа №7.-
https://docs.google.com/document/d/1itG2cWyRZuS1kcfdiKPSxGa8oqEYpKOJ0Zow9xZN9PA/edit?usp=sharing

Практическая работа№8-
https://docs.google.com/document/d/1bEeC_-gimtQSE7an3zliCuADoctnsD6gHQREm_xAySI/edit?usp=sharing

Практическая работа№9-
https://docs.google.com/document/d/1J8RmnDmq97gH-v2Hr6Vlf7orKEbQIa0tDo5YwWIFJ4c/edit?usp=sharing

Практическая работа№10-
https://docs.google.com/document/d/13I3XSjgW3QRm2N7Ok9w3BWASP-rfQStSkkOxbRZQJS0/edit?usp=sharing

Практическая работа№11-
https://docs.google.com/document/d/1rSSvodYvJrSjgdwP6-lsm5a8z_HinNL_H7kH50Ft1vg/edit?usp=sharing

Практическая работа№12-
https://docs.google.com/document/d/1cfAmcXnhjdSsCh28-IOPefkSTLXaukp-sdFv1U5s8Pk/edit?usp=sharing