IT CAREER HUB

Урок 10.
Знакомство с коллекциями: списки

Итерируемый объект	**2**
Цикл for	**3**
Задания для закрепления	**5**
Функция range	**6**
Задания для закрепления	**9**
Операторы break, continue, else в цикле for	**11**
Задания для закрепления	**14**
Вложенные циклы	**16**
Вложенные циклы с использованием while и for	**18**
Задания для закрепления	**19**
Практические задания	**20**

IT CAREER HUB

Коллекции

Коллекции — это структуры данных, которые позволяют хранить и организовывать множество элементов.

В Python существует несколько встроенных типов коллекций, каждая из которых имеет свои особенности и подходит для различных задач.

IT CAREER HUB

Изменяемые типы данных

Изменяемые типы данных — это такие структуры данных, которые позволяют изменять своё содержимое после создания объекта.

Это означает, что можно добавлять, удалять или изменять элементы в существующем объекте, не создавая при этом новый объект.

Список (list)

Список (list) — это упорядоченная изменяемая (mutable) коллекция, которая может содержать элементы любых типов.

Список позволяет добавлять, удалять и изменять элементы, а также хранить их в определённом порядке. Это одна из самых универсальных и часто используемых структур данных в Python.

Основные характеристики списка:

1. **Изменяемость:** Элементы списка можно изменять после его создания.
2. **Упорядоченность:** Элементы списка хранятся в том порядке, в котором они были добавлены.
3. **Поддержка дубликатов:** Список может содержать повторяющиеся элементы.
4. **Поддержка различных типов данных:** Список может содержать элементы разных типов.
5. **Индексация:** К элементам списка можно обращаться по индексу.

Пример создания списка

```python
fruits = ["apple", "banana", "cherry"]  # Список со строками

print(fruits)

numbers = [1, 2, 3]  # Список с числами

print(numbers)
```

Хранение разных типов элементов в списке

Список (`list`) является универсальной структурой данных, которая может хранить элементы разных типов данных. Это одно из главных преимуществ списков в Python: в одном списке могут находиться строки, числа, логические значения, другие списки и даже функции.

IT CAREER HUB

Пример списка с элементами разных типов

```python
mixed_list = [42, "Python", 3.14, True, [1, 2, 3]]

print(mixed_list)
```

В этом примере:

- 42 — это целое число.
- `"Python"` — это строка.
- `3.14` — это число с плавающей точкой (вещественное число).
- `True` — это логическое значение.
- `[1, 2, 3]` — это вложенный список.

Создание списков

- **Создание списка с элементами**
  Список создаётся с использованием квадратных скобок `[]`, а элементы внутри разделяются запятыми.

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

print(my_list)
```

- **Создание пустого списка**
  - Для создания пустого списка используются квадратные скобки без элементов.

```python
empty_list = []

print(empty_list)
```

- Также пустой список можно создать с помощью функции `list()`.

```python
my_list = list()

print(my_list)
```

IT CAREER HUB

Задания для закрепления 1

1. **Что можно хранить в списке в Python?**

a. Только строки
b. Только числа
c. Только числа и строки
d. Элементы разных типов данных

Посмотреть ответ

2. Какой результат будет выведен при выполнении следующего кода?

```python
my_list = list()

print(my_list)
```

a. `[]`
b. `[None]`
c. `list()`
d. `Ошибка`

Посмотреть ответ

IT CAREER HUB

Индексация списков

Индексация — это процесс доступа к элементам списка по их позиции (индексу) внутри списка.

Индексация, как и в строках, начинается с нуля для первого элемента и идёт последовательно для следующих элементов. Используя индексы, можно не только получить доступ к отдельным элементам, но изменить их или удалить.

Пример индексации

```python
my_list = ["apple", "banana", "cherry", "date"]

# Доступ к первому элементу
print(my_list[0])

# Доступ к последнему элементу с положительным индексом
print(my_list[3])

# Доступ к последнему элементу с отрицательным индексом
print(my_list[-1])

# Доступ к первому элементу с отрицательным индексом
print(my_list[-4])
```

IT CAREER HUB

Срезы списков

Срезы (slices) — это способ получения части списка (подсписка), без изменения исходного списка. Срезы позволяют выбрать элементы по индексу с указанием начала, конца и шага.

Основной синтаксис срезов:

```python
list[start:stop:step]
```

- **start (необязательно)** — индекс начала среза (включительно). По умолчанию — 0.
- **stop (обязательно)** — индекс конца среза (не включительно).
- **step (необязательно)** — шаг, с которым нужно выбирать элементы. По умолчанию — 1.

Примеры использования срезов

Срезы положительным индексом

```python
my_list = [0, 1, 2, 3, 4, 5, 6]

# Срез с 1-го по 4-й элемент (не включительно)
print(my_list[1:4])

# Срез каждого второго элемента
print(my_list[::2])

# Срез каждого второго элемента в обратном порядке
print(my_list[::-2])
```

```python
# Срез от 4-го элемента до начала
print(my_list[4::-1])

# Срез с 3-го элемента до конца
print(my_list[3:])

# Копия списка с помощью среза
print(my_list[:])
```

Срезы отрицательным индексом

Отрицательные индексы позволяют отсчитывать элементы с конца списка.

Python
```python
my_list = [0, 1, 2, 3, 4, 5, 6]

# Срез от -4-го до -1-го элемента
print(my_list[-4:-1])

# Срез от -2-го до -5-го элемента
print(my_list[-2:-5:-1])

# Срез списка в обратном порядке
print(my_list[::-1])
```

Изменение по индексу

В Python элементы списка можно изменять напрямую, используя индексацию.

Это одна из ключевых особенностей списков — они являются изменяемыми (mutable), что позволяет легко модифицировать содержимое списка, обращаясь к конкретным элементам по их индексу.

Пример изменения элементов

Вы можете обратиться к элементу списка по его индексу и присвоить ему новое значение.

```python
my_list = ["apple", "banana", "cherry"]



# Изменим второй элемент (с индексом 1)

my_list[1] = "blueberry"

print(my_list)
```

В этом примере элемент с индексом 1 (второй элемент списка) был заменён на "blueberry".

Изменение по срезу

Можно изменить сразу несколько элементов списка, используя индексацию срезов.

```python
my_list = [10, 20, 30, 40, 50]
```

```
# Заменим второй и третий элементы
my_list[1:3] = [200, 300]
print(my_list)
```

Использование отрицательных индексов для изменения

Отрицательные индексы можно использовать для изменения элементов списка с конца.

```
Python
my_list = [100, 200, 300, 400]


# Заменим последний элемент

my_list[-1] = 500

print(my_list)
```

IT CAREER HUB

Задания для закрепления 2

1. Какой результат будет выведен при выполнении следующего кода?

```python
my_list = [1, 2, 3, 4]
print(my_list[2])
```

a. 1
b. 2
c. 3
d. 4

Посмотреть ответ

2. Какой результат будет выведен при выполнении следующего кода?

```python
my_list = [0, 1, 2, 3, 4, 5]
print(my_list[::-1])
```

a. [0, 1, 2, 3, 4, 5]
b. [5, 4, 3, 2, 1, 0]
c. [0, -1, -2, -3, -4, -5]
d. Ошибка

Посмотреть ответ

3. Какое значение будет в списке my_list после выполнения следующего кода?

```python
my_list = ["a", "b", "c", "d"]
my_list[1] = 0
print(my_list)
```

a. ['a', 'b', 'c', 'd']
b. ['a', 0, 'c', 'd']
c. ['0', 'b', 'c', 'd']
d. ['a', 0, 'c', 'd']

Посмотреть ответ

IT CAREER HUB

Операции со списками

Списки в Python поддерживают множество операций, позволяющих изменять, объединять или проверять списки. Ниже приведены основные операции со списками.

Конкатенация списков (объединение)

Списки можно объединять с помощью оператора +. Эта операция создаёт новый список, объединяя два или более списков.

```python
list1 = [1, 2, 3]

list2 = [4, 5, 6]

combined_list = list1 + list2

print(combined_list)
```

Повторение списка

Список можно повторить несколько раз с помощью оператора *. Эта операция создаёт новый список, состоящий из повторённых элементов исходного списка.

```python
repeated_list = [0] * 5

print(repeated_list)

my_list = [1, 2, 3]

repeated_list = my_list * 3

print(repeated_list)
```

Проверка на наличие элемента в списке

С помощью оператора `in` можно проверить, содержится ли определённый элемент в списке.

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

print(3 in my_list)

print(6 in my_list)

my_list = ["apple", "banana", "cherry"]

print("apple" in my_list)

print("app" in my_list)  # Ищет полное совпадение элемента
```

Длина списка

Функция `len()` возвращает количество элементов в списке.

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

print(len(my_list))
```

Преобразование в список

Можно преобразовать строку или другую коллекцию в список с помощью функции `list()`.

- **Преобразование из строки:**

```python
word = "python"

my_list = list(wor # Строка "разбивается" на отдельные символы
```

```
print(my_list)

print(type(my_list))
```

- **Преобразование из объекта range:**

```
Python
my_range = range(1, 11, 2)

print(my_range)

print(type(my_range))

my_list = list(my_range)

print(my_list)
```

Сравнение списков

Списки можно сравнивать друг с другом с помощью операторов сравнения.

Сравнение списков выполняется поэлементно, начиная с первого элемента, и продолжается до тех пор, пока не будет найдено различие или пока все элементы не будут проверены.

Примеры

```
Python
list1 = [1, 2, 3]
list2 = [1, 2, 3]
list3 = [1, 3, 0]

# Сравнение на равенство
```

```
print(list1 == list2)
print(list1 == list3)


# Сравнение на неравенство
print(list1 != list2)
print(list1 != list3)

### Сравнение на больше/меньше, больше или равно/меньше или равно
print(list1 < list2)
print(list3 > list1)
print(list1 <= list2)
print(list1 >= list3)
```

Поддержка операции сравнения между типами

Для корректного сравнения списков, элементы должны быть одного типа или поддерживать операцию сравнения между собой.

Например, числа можно сравнивать с числами, строки — со строками. Если в списках присутствуют несравнимые типы данных (например, число и строка), это приведёт к ошибке `TypeError`.

Пример корректного сравнения

```python
list1 = [1, 2.6, 2]
list2 = [1, 2, 3]

# int и float можно сравнивать между собой
print(list1 < list2)
```

Пример ошибки типов

```python
list1 = [1, 2, "apple"]

list2 = [1, 2, 3]



# Попытка сравнить числовое значение с строкой вызовет ошибку TypeError

print(list1 < list2)
```

```
Traceback (most recent call last): Explain with AI
  File "/home/tanya/PycharmProjects/pythonProgramItch/_notes/test.py", line 5, in <module>
    print(list1 < list2)
          ^^^^^^^^^^^^^
TypeError: '<' not supported between instances of 'str' and 'int'

Process finished with exit code 1
```

Особенности поэлементного сравнения

Важно учитывать, что первом нахождении отличающихся элементов интерпретатор остановит сравнение. Поэтому если после них будут находиться несравниваемые типы это не приведет к ошибке.

Пример

```python
list1 = [1, 2, "apple"]
list2 = [1, 3, 10]

print(list1 < list2)  # Вернёт True, так как 2 < 3, сравнение остановится до строки "apple"
```

IT CAREER HUB

Задания для закрепления 3

1. Какой результат будет выведен при выполнении следующего кода?

```python
list1 = [1, 2, 3]

list2 = [4, 5, 6]

combined_list = list1 + list2

print(combined_list)
```

a. [1, 2, 3, 4, 5, 6]
b. [4, 5, 6, 1, 2, 3]
c. [1, 2, 3], [4, 5, 6]
d. Ошибка

Посмотреть ответ

2. Какой результат будет выведен при выполнении следующего кода?

```python
my_list = [0] * 4

print(my_list)
```

a. [4]
b. [1, 1, 1, 1]
c. [0]
d. [0, 0, 0, 0]

Посмотреть ответ

IT CAREER HUB

3. Какой результат будет выведен при выполнении следующего кода?

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

print(6 in my_list)
```

a. True
b. False
c. Ошибка
d. Ничего не выведется

Посмотреть ответ

IT CAREER HUB

Цикл for со списками

С помощью цикла `for` можно последовательно проходить через элементы коллекций. Python автоматически перебирает элементы коллекции по порядку, упрощая обработку данных.

Пример

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


for item in my_list:

    print(item)
```

Вложенный цикл for со списками

Вложенные циклы `for` позволяют перебирать элементы списка, где каждый элемент сам по себе является итерируемым объектом, например, строкой. Внутренний цикл проходит по символам каждой строки.

Пример: вложенный цикл for по списку строк

```python
my_strings = ["apple", "banana", "cherry"]

for word in my_strings:
    for letter in word:
        print("letter:", letter)
    print()
```

IT CAREER HUB

Задания для закрепления 4

1. **Какой результат будет выведен при выполнении следующего кода?**

```python
my_strings = ["cat", "dog"]

for word in my_strings:

    for letter in word:

        print(letter, end="")

    print()
```

a. c a t d o g
b. catdog
c. c a t d o g
d. cat dog

IT CAREER HUB

Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
1. Списки в Python	Ответ: d
2. Результат выполнения кода	Ответ: a
Задания на закрепление 2	Вернуться к заданиям
1. Результат выполнения кода	Ответ: c
2. Результат выполнения кода	Ответ: b
3. Результат выполнения кода	Ответ: b
Задания на закрепление 3	Вернуться к заданиям
1. Результат выполнения кода	Ответ: a
2. Результат выполнения кода	Ответ: d
3. Результат выполнения кода	Ответ: b
Задания на закрепление 4	Вернуться к заданиям
Результат выполнения кода	Ответ: d

IT CAREER HUB

Практические задания

1. **Напишите программу, которая выводит все числа от 1 до n (введённого пользователем), которые делятся на 3.**

Пример вывода:

```
Python
Введите число: 15
3
6
9
12
15
```

Решение:

```python
n = int(input("Введите число: "))
numbers = list(range(0, n + 1, 3))

for num in numbers:
    print(num)
```

IT CAREER HUB

2. **Напишите программу, которая обрабатывает список строк, состоящий из имён. Нужно вывести только те имена, длина которых больше средней длины имен в списке.**

Пример данных для обработки:

```python
names = ["John", "Bob", "Alice", "Anna", "Mark"]
```

Пример вывода:

```
Список имён: ['John', 'Bob', 'Alice', 'Anna', 'Mark']
Средняя длина имён: 4.0
Имена длиннее средней длины:
Alice
```

Решение:

```python
names = ["John", "Bob", "Alice", "Anna", "Mark"]

# Рассчитаем среднюю длину имён
sum_len = 0
for name in names:
    sum_len += len(name)
average_length = sum_len / len(names)


# Выводим имена, длина которых больше средней
print("Список имён:", names)
print("Средняя длина имён:", average_length)
print("Имена длиннее средней длины:")
for name in names:
    if len(name) > average_length:
        print(name)
```