Университет Сириус Колледж

Основы программирования
Лекция 3.

Циклы. Кортежи. Словари. Множества.

Университет Сириус Колледж

Цикл while

Инструкция `while` в Python повторяет указанный блок кода до тех пор, пока указанное в цикле условие будет оставаться истинным.

```
while <условие>:
    <выражение>
```

Университет Сириус Колледж

Цикл while

Инструкция `while` в Python повторяет указанный блок кода до тех пор, пока указанное в цикле условие будет оставаться истинным.

```
apples = 5

while apples > 0:
    print('We have', apples, 'apples')
    apples -= 1

print('We have no apples any more!')
```

Университет Сириус Колледж

Цикл for

Этот цикл проходится по любому итерируемому объекту (например, строке или списку), и во время каждого прохода выполняет тело цикла.

```
1  for <элемент> in <последовательность>:
2      <выражение>
```

Цикл for

Этот цикл проходится по любому итерируемому объекту (например, строке или списку), и во время каждого прохода выполняет тело цикла.

```
1  for i in range(5):
2      print(i)
```

Университет Сириус Колледж

Цикл for

Этот цикл проходится по любому итерируемому объекту (например, строке или списку), и во время каждого прохода выполняет тело цикла.

```
1  word = 'Слово'
2
3  for letter in word:
4      print(letter)
```

Университет Сириус Колледж

Цикл for

Этот цикл проходится по любому итерируемому объекту (например, строке или списку), и во время каждого прохода выполняет тело цикла.

```
1  shopping_list = ['milk', 'bread', 'cucumber', 'butter']
2
3  for element in shopping_list:
4      print(element)
```

Университет Сириус Колледж

Цикл for

Этот цикл проходится по любому итерируемому объекту (например, строке или списку), и во время каждого прохода выполняет тело цикла.

```
1  seasons = {1: 'Зима', 2: 'Весна', 3: 'Лето', 4: 'Осень'}
2
3  for s in seasons:
4      print('Номер сезона:', s)
5      print('Название:', seasons[s])
```

Университет Сириус Колледж

Оператор `continue`

Оператор `continue` начинает следующий проход цикла, минуя оставшееся тело цикла (`for` или `while`). Пример:

```
1  for letter in 'Стол':
2      if letter == 'т':
3          continue
4      print(letter)
```

Результат выполнения:

```
С
о
л
```

Университет Сириус Колледж

Оператор `break`

Оператор `break` досрочно прерывает цикл.

```
for letter in 'Стол':
    if letter == 'т':
      break
    print(letter)
```

Результат выполнения:

```
С
```

Университет Сириус Колледж

`else` в циклах

Циклы `for` и `while` в Python имеют свою конструкцию `else`, которая выполняется после завершения цикла.

```
1  for letter in 'Стол':
2      print(letter)
3  else:
4      print("Цикл завершился нормально")
```

```
С
т
о
л
Цикл завершился нормально
```

Университет Сириус Колледж

`else` в циклах

Циклы `for` и `while` в Python имеют свою конструкцию `else`, которая выполняется после завершения цикла.

```
for letter in 'Стол':
    if letter == 'т':
        continue
    print(letter)
else:
    print("Цикл завершился нормально")
```

```
С
т
о
л
Цикл завершился нормально
```

Университет Сириус Колледж

`else` в циклах

Циклы `for` и `while` в Python имеют свою конструкцию `else`, которая выполняется после завершения цикла.

```
for letter in 'Стол':
    if letter == 'т':
        break
    print(letter)
else:
    print("Цикл завершился нормально")
```

С

Университет Сириус Колледж

Функция `range()`

Функция `range()` генерирует последовательность чисел, которую часто используют для итерации в циклах `for`. Она может принимать от одного до трёх аргументов: `start`, `stop` и `step`.

```
1  for i in range(5):  # Генерирует числа от 0 до 4
2      print(i)
```

```
0
1
2
3
4
```

Университет Сириус Колледж

Функция `range()`

Функция `range()` генерирует последовательность чисел, которую часто используют для итерации в циклах `for`. Она может принимать от одного до трёх аргументов: `start`, `stop` и `step`.

```
1  for i in range(2, 10):  # Генерирует числа от 2 до 9
2      print(i)
```

```
2
3
4
5
6
7
8
9
```

Университет Сириус Колледж

Функция `range()`

Функция `range()` генерирует последовательность чисел, которую часто используют для итерации в циклах `for`. Она может принимать от одного до трёх аргументов: `start`, `stop` и `step`.

```
1  for i in range(1, 10, 2):  # Генерирует нечётные числа от 1 до 9
2      print(i)
```

```
1
3
5
7
9
```

Университет Сириус Колледж

Функция `enumerate()`

Функция `enumerate()` добавляет счётчик к итерируемому объекту и возвращает его в виде объекта перечисления. Это полезно, когда нужно получить индекс элемента вместе с самим элементом при итерации.

```
1  words = ['apple', 'banana', 'cherry']
2
3  for index, word in enumerate(words):
4      print(index, word)
```

Университет Сириус Колледж

Функция zip()

Функция zip() принимает несколько итерируемых объектов и возвращает итератор, который генерирует кортежи, состоящие из элементов переданных итерируемых объектов, взятых по одному из каждого.

```
names = ['Анна', 'Борис', 'Виктор']
ages = [25, 30, 35]

for name, age in zip(names, ages):
    print(name, age)
```

Университет Сириус Колледж

Кортежи (tuple)

- Кортежи служат для хранения нескольких объектов вместе. Их можно рассматривать как аналог списков, но без такой обширной функциональности, которую предоставляет класс списка.
- Одна из важнейших особенностей кортежей заключается в том, что они неизменяемы.
- Кортежи обозначаются указанием элементов, разделённых запятыми; по желанию их можно ещё заключить в круглые скобки.
- Кортежи обычно используются в тех случаях, когда оператор или пользовательская функция должны наверняка знать, что набор значений, т.е. кортеж значений, не изменится.
- Элементы кортежа могут быть любого неизменяемого типа данных, включая числа, строки и даже другие кортежи.

Университет Сириус Колледж

Примеры кортежей

```
1  zoo = ('python', 'elephant', 'pinguin')
2  new_zoo = 'monkey', 'cheetah'
```

Несмотря на то, что кортежи можно объявлять и использовать без скобок, является хорошей практикой обособлять их скобками.

> *Явное лучше неявного.*

Университет Сириус Колледж

Особенности минимальных кортежей

- Пустой кортеж объявляется очень просто - с помощью пустых скобок `()`.
- Кортеж из одного элемента объявляется с помощью конструкции скобок, элемента в них, а также запятой после этого единственного элемента, вот так:

```
1  (2,)
```

Так интерпретатор Python поймёт, что вы не просто заключили какое-то значение в скобки, а именно определяете кортеж.

Университет Сириус Колледж

Особенности использования кортежей

- Передавая кортеж функции, необходимо дублировать скобки. Например, эти две строки кода выполняют разные задачи:

```
1  print(1, 2, 3)
```

и

```
1  print( (1, 2, 3) )
```

- Первый код отправит в стандартный вывод значения `1, 2, 3`.
- Второй код отправит в него кортеж `(1, 2, 3)`. Внешние скобки заключают аргументы, передаваемые функции `print`. Внутренние заключают кортеж.

Университет Сириус Колледж

Словари (dictionaries)

- Словарь — это структура данных, в которой уникальные ключи (имена) связаны со значениями (информацией).
- Заметьте, что ключ должен быть именно уникальным, и это обязательно.
- Также важно, что в качестве ключей могут служить только неизменяемые (хешируемые) типы данных, в то время как в качестве значений можно использовать как изменяемые, так и неизменяемые типы данных.

Университет Сириус Колледж

Создание и инициализация словарей

```
# Пустой словарь
>> my_dict = {}
{}
>> my_dict = dict()
{}
```

Университет Сириус Колледж

Создание и инициализация словарей

```
1  # Словарь с начальными значениями
2  >> student = {'group' : '09.02.07', 'age' : 18 }
3  {'group' : '09.02.07', 'age' : 18 }
```

Университет Сириус Колледж

Создание и инициализация словарей

```
# Словарь с начальными значениями
>> student = dict(group='09.02.07', age=18)
{'group' : '09.02.07', 'age' : 18 }
```

Университет Сириус Колледж

Создание и инициализация словарей

```
# Словарь с начальными значениями
>> student = dict([('group', '09.02.07'), ('age', 18)])
{'group' : '09.02.07', 'age' : 18 }
```

Университет Сириус Колледж

Создание и инициализация словарей

```
# Dictionary comprehension
>> student = {k: v for k, v in [('group', '09.02.07'), ('age', 18)]}
{'group' : '09.02.07', 'age' : 18 }
```

Университет Сириус Колледж

Создание и инициализация словарей

```
# Dictionary comprehension
>> student = {k: v for k, v in zip(['group', 'age'], ['09.02.07', 18])}
{'group' : '09.02.07', 'age' : 18 }
```

Университет Сириус Колледж

Создание и инициализация словарей

```
# Dictionary comprehension
>> student = {k: v for k, v in enumerate(['09.02.07', 18])}
{0: '09.02.07', 1: 18 }
```

Университет Сириус Колледж

Создание и инициализация словарей

```
# из списка ключей
>> keys = ['group', 'age']
>> dict.fromkeys(keys)
{'group' : None, 'age' : None }
```

Университет Сириус Колледж

Создание и инициализация словарей

```
# из списка ключей с одним значением по умолчанию
>> keys = ['group', 'age']
>> dict.fromkeys(keys, 18)
{'group' : 18, 'age' : 18 }
```

Университет Сириус Колледж

Примеры словарей

```
1  student_1 = {'group' : '09.02.01', 'age' : 18 }
2  student_2 = {'group' : '09.02.06', 'age' : 18 }
```

Университет Сириус Колледж

Примеры словарей

```
1  student_1 = {'group' : '09.02.01', 'age' : 18 }
2  student_2 = {'group' : '09.02.06', 'age' : 18 }
```

Получение значения по ключу словаря использует конструкцию `имя_словаря[ключ]`:

```
1  student_1['age'] == 18
```

Университет Сириус Колледж

Примеры словарей

```
student_1 = {'group' : '09.02.01', 'age' : 18 }
student_2 = {'group' : '09.02.06', 'age' : 18 }
```

Получение значения по ключу словаря использует конструкцию `имя_словаря[ключ]`:

```
student_1['age'] == 18
```

Так же можно использовать метод `get()` для получения значения по ключу:

```
student_1.get('age') == 18
```

Университет Сириус Колледж

Примеры словарей

```
student_1 = {'group' : '09.02.01', 'age' : 18 }
student_2 = {'group' : '09.02.06', 'age' : 18 }
```

Получение значения по ключу словаря использует конструкцию `имя_словаря[ключ]`:

```
student_1['age'] == 18
```

Так же можно использовать метод `get()` для получения значения по ключу:

```
student_1.get('age') == 18
```

В случае, если ключ отсутствует в словаре, метод `get()` вернёт `None` или значение по умолчанию, если оно указано:

```
>> student_1.get('course', "no course")
"no course"
```

Университет Сириус Колледж

Примеры словарей

В качестве значений словарь может использовать другие словари:

```
students = {'Ivan' : student_1, 'Petr' : student_2}
```

Таким образом:

```
{'Ivan' : {'group' : '09.02.01', 'age' : 18 },
 'Petr' : {'group' : '09.02.06', 'age' : 18 }}
```

Получить значение из вложенного словаря можно получить с помощью конструкции `имя_словаря[ключ_1][ключ_2]`:

```
students['Ivan']['group'] == '09.02.01'
```

Университет Сириус Колледж

Методы keys(), values(), items()

Вывести список ключей словаря можно с помощью метода `keys()`:

```
students.keys() == ['Ivan', 'Petr]
```

Вывести список значений словаря можно с помощью метода `values()`:

```
student_1.values() == ['09.02.01', 18]
```

Вывести список кортежей словаря "ключ-значение" можно с помощью метода `items()`:

```
student1.items() == [
    ('group', '09.02.01'),
    ('age', 18)
    ]
```

Университет Сириус Колледж

Множество (set)

- Множество - это структура данных, содержащая уникальные элементы в случайном порядке.
- Создать пустое множество можно с помощью фигурных скобок `{}` или с помощью функции `set()`.
- Примечательно, что если передать функции `set()` любой итерируемый тип данных с неуникальными элементами, то эта функция вернёт множество уникальных элементов из переданных ей.

```
1  set(['student', 'professor', 'student']) == {'student', 'professor'}
2  set('hello') == {'h', 'e', 'l', 'o'}
```

Университет Сириус Колледж

Операции с множествами

- `len(s)` - число элементов в множестве (размер множества).
- `x in s` - принадлежит ли `x` множеству `s`.
- `set.isdisjoint(other)` - истина, если `set` и other не имеют общих элементов.
- `set == other` - все элементы set принадлежат other, все элементы other принадлежат `set`.
- `set.issubset(other)` или `set ⩽ other` - все элементы `set` принадлежат other.
- `set.issuperset(other)` или `set ⩾ other` - аналогично.
- `set.union(other, ...)` или `set | other | ...` - объединение нескольких множеств.
- `set.intersection(other, ...)` или `set & other & ...` - пересечение.
- `set.difference(other, ...)` или `set - other - ...` - множество из всех элементов `set`, не принадлежащие ни одному из `other`.

Университет Сириус Колледж

Операции с множествами

- `set.symmetric_difference(other)`; `set ^ other` - множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.
- `set.update(other, ...)`; `set |= other | ...` - объединение.
- `set.intersection_update(other, ...)`; `set &= other & ...` - пересечение.
- `set.difference_update(other, ...)`; `set -= other | ...` - вычитание.
- `set.symmetric_difference_update(other)`; `set ^= other` - множество из элементов, встречающихся в одном множестве, но не встречающиеся в обоих.
- `set.add(elem)` - добавляет элемент в множество.
- `set.remove(elem)` - удаляет элемент из множества. `KeyError`, если такого элемента не существует.
- `set.discard(elem)` - удаляет элемент, если он находится в множестве.
- `set.pop()` - удаляет первый элемент из множества. Так как множества не упорядочены, нельзя точно сказать, какой элемент будет первым.
- `set.clear()` - очистка множества.

Университет Сириус Колледж

frozenset

Единственное отличие `set` от `frozenset` заключается в том, что `set` - изменяемый тип данных, а `frozenset` - нет. Примерно похожая ситуация с списками и кортежами.

```
1  a = set('qwerty')
2  b = frozenset('qwerty')
3  a.add(1)
4  b.add(1)
```

```
Traceback (most recent call last):
File "<stdin>", line 4, in <module>
AttributeError: 'frozenset' object has no attribute 'add'
```