ТЕХНОЛОГИЧЕСКАЯ КАРТА ЗАНЯТИЯ

Тема занятия: Циклы

Аннотация к занятию: в первой части урока учащиеся изучают применение индексов и срезов доступа к части строки, учатся использовать функции и методы строк для решения задач. Обучающиеся знакомятся с циклическим алгоритмом с заданным условием продолжения работы, изучают синтаксис цикла while и соотносят его с блок-схемой. Также обучающиеся знакомятся с циклом for. Вторая часть урока посвящена решению задач.

Цель занятия: формирование знаний о циклах и решение задач на циклы на языке программирования Python.

Задачи занятия:

- научить обращаться к символу строки по индексу;
- научить применять срезы для извлечения части строки;
- познакомить с функциями и методами для работы со строками;
- познакомить с циклом while;
- научить применять цикл с условием для решения разнообразных задач;
- познакомить с циклом for;
- научить управлять циклом с помощью операторов break и continue;
- научить применять цикл с параметром for для решения разнообразных задач.

Ход занятия

Этап занятия	Время	Деятельность педагога	Комментарии, рекомендации для педагогов
Организационный этап	2 мин.	Добрый день! Как ваше настроение?	
Постановка цели и задач занятия. Мотивация учебной деятельности обучающихся	10 мин.	На прошлых занятиях мы познакомились с операторами сравнения и новым типом данных — bool. Всё это — небольшие кирпичики в большой теме логического ветвления. Ранее программы, которые мы создавали, умели считывать и выводить значения, но их алгоритмы оставались линейными. Вне зависимости от полученной информации был только один итог. Вопрос для обсуждения: Всегда ли мы при выполнении алгоритма движемся линейно? Возможный ответ учеников: Иногда нам нужно принимать решение об изменении маршрута движения. Вопрос для обсуждения: Как звучит в естественном языке разветвление при условии? Например, при принятии решения о том, брать ли зонт. Возможные ответы учеников: Если ..., то ... . Если на улице дождь, то я возьму зонт.	Обсуждается необходимость применения механик, которые бы позволяли разветвлять линейный алгоритм при решении задач.

		Вопрос для обсуждения: Бывают ли какие-то процессы, которые повторяются много раз? Назовите примеры. **Возможный ответ учеников:** Да. Например, смена дня и ночи. **Вопрос для обсуждения:** Как будет звучать в описании решения некой задачи, что мы должны что-то делать до выполнения какого-то условия? **Возможный ответ учеников:** До тех пор, пока **Вопрос для обсуждения:** Представьте, что вам нужно открыть 5 ящиков и выложить всё из них. Озвучьте, как это будет выглядеть. **Возможный ответ учеников:** Я подхожу к первому ящику. Вытаскиваю оттуда вещи. Иду ко второму … И так до пятого.	
Изучение нового материала	45 мин.	1. Операции над строками Инициализация Тип переменной строки (str) необходим для хранения текста. По умолчанию используется кодировка стандарта UTF-8. Задавать строку можно одинарными и двойными кавычками. Важно, чтобы в начале и конце они были одинаковыми. Если в строку нужно добавить сам символ одинарной или двойной кавычки, это можно сделать с помощью слэша, как показано в примере.	Ссылка на Google Colab: https://colab.research.google.com/drive/1saOgLI4VtDgHkzOuImOATygTlyBQPTNo?usp=sharing

```
[ ]  1  x = "abc"
     2  y = 'xyz'
     3  print(x, '|', type(x))
     4  print(y, '|', type(y))
```

```
abc | <class 'str'>
xyz | <class 'str'>
```

```
[ ]  1  'экранирование символа ковычки \''
```

```
'экранирование символа ковычки ''
```

Операции
Со строками можно выполнять разные операции. Например, конкатенацию. В примере фраза «Привет, Мир» записалась 4 раза.

```
[ ]  1  a = 'Привет'
     2  b = "Мир"
     3  s = a + " " + b
     4  print(s)
```

```
Привет Мир
```

Или умножить:

```
[ ]  1  a * 4
```

```
'ПриветПриветПриветПривет'
```

Как и во многих языках программирования, в Python есть возможность ввода с клавиатуры с помощью функции input(). В примере функция сохраняется в переменную a. Конечно, если вам нужно использовать числовую запись, нужно явным образом вызывать преобразование типов. По умолчанию будет возвращаться строка.

```
[ ]   1   a = input('Введите что-то: ')
      2   type(a)
```

```
Введите что-то:123
str
```

Хранение
Не секрет, что строки — это комбинация букв. А вот для хранения букв и, как следствие, строк используют числа. За то, какому символу будет присвоено определённое значение, отвечает таблица «Аски-2» (ASCII). В таблице всего 256 символов, но этого хватает, чтобы охватить весь латинский, кириллический алфавит и ряд специальных символов: начало и конец файла, перенос строки, пробелы и прочее. Их значения меньше 32, и они не отображаются на экране.

Чтобы узнать кодировку символа, понадобится функция ord. Важно: ord принимает на вход на только один символ.

chr — функция, которая по номеру в таблице возвращает соответствующий символ.

```
1  chr(78)
```

```
'N'
```

```
[ ]  1  chr(166)
```

```
'¦'
```

```
[ ]  1  ord('a')
```

```
97
```

Таким образом, используя функцию ord, можно сравнивать символы и последовательности символов, выстраивая их в лексикографическом порядке. То есть как в словаре. Такие вещи важны, например, в биоинформатике, при работе с суффиксными массивами.

Сравнение символов и строк
Для определения лексикографического порядка строк разной длины нужно мысленно дописать символы с кодом в конец короткой строки. Код должен быть меньше кода любого возможного символа. Затем применяем стандартное правило. Подробнее о сравнении вы можете прочитать по ссылке.

```
1  ord('a'), ord('b')
```

```
(97, 98)
```

+ Код + Текст

```
1  'b' > 'a'
```

```
True
```

Индексация

Индексация и слайсинг — важные понятия этого урока. Их синтаксис идентичен и для других итерируемых структур, которые мы увидим далее.

Поскольку строка — это последовательность символов, мы можем обратиться к их позиции. Отсчёт при индексации в Python всегда начинается с нуля. В примере под нулевым индексом строки а большая буква П. Чтобы обратиться к ней, мы открываем квадратные скобки и записывает туда индекс. Это целое число можно подать в скобки из какой-то внешней функции, ищущей по строкам.

```
[ ]  1  a = 'Привет'
     2
     3  a[0]
```

```
'П'
```

```
[ ]  1  a[1]
```

```
'р'
```

Индексацию можно вести и по отрицательным числам. Минус первый символ — это последний, минус второй — предпоследний, и так далее.

```
[ ]  1  # отобразим предпоследний символ
     2  a[-2]
```

```
'е'
```

Однако выйти за пределы мы не можем: 5-й индекс в слове «привет» это буква т, а 6-й уже не определён. Появляется ошибка IndexError.

```
1  # 5й индекс в строке еще определен (буквой т), а 6й - уже нет
2  a[6]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-34-5d305e7d1547> in <module>()
      1 # 5й индекс в строке еще определен (буквой т), а 6й - уже нет
----> 2 a[6]

IndexError: string index out of range
```

SEARCH STACK OVERFLOW

Слайсинг или сечение

Часто нам важно выделить небольшой диапазон значений, в котором заключена полезная информация. Для этого нам нужен слайсинг или срез, он же сечение.

Происходит он аналогично индексации: вы пишете строку или переменную, содержащую в себе строку, и открываете рядом с ним квадратные скобки. Здесь дано формальное определение, однако проще запомнить его так. Внутри квадратных скобок через двоеточие перечислены три значения: n, m, k.

N — с какого индекса включительно мы начинаем срез, m — по какой индекс, не включая его, k — с каким шагом отображать результат. Разделяет эти целые числа двоеточие.

Если не указывать ничего, т.е. просто оставить квадратные скобки и два двоеточия, мы отобразим всю строку с первого по последний элемент включительно с шагом один. Если не

указывать n (первый параметр), Python заполнит его 0. Если не указывать второй параметр m, Python заполнит его длиной строки — тем, что возвращает нам встроенная функция len. Если не указать третий аргумент k, Python заполнит его единицей.

$$s[n:m:k]$$

Рассмотрим это на примере строки abcd. Это 4 символа, записанные в переменную a, с индексами с нулевого по третий. Делая срез с нулевого индекса включительно по третий индекс, не включая его, мы отобразим буквы с индексами 0, 1, 2, то есть abc.

```
1  a = 'abcd'
2
3  # будет отображены буква с индексами начиная с нуля включая по индекс буквы на позиц
4  a[0:3]
```

```
'abc'
```

Ещё один пример среза с нулевого индекса включительно по четвёртый индекс, не включая его, с шагом 2. Так мы отобразим индексы 0 и 2, то есть буквы a и c.

```
1  # с нулевой включая по 4ю не включая, с шагом 2 (по умолчанию шаг 1)
2  a[0:4:2]
```

```
'ac'
```

В этом примере мы проводим слайсинг с первого индекса включительно и не указываем второй аргумент. Значит, до len(a). Отображаем 1, 2 и 3 индексы — значения b, c, d.

```
# второй аргумент не указан, по умолчанию с первого индекса
a[1:]
```

```
'bcd'
```

В следующем примере не указан первый аргумент. Значит, мы начнём с нулевого по 2-й с шагом один, который тоже не указан:

```
# первый аргумент не указан, по умолчнию с начала 0 индекса
a[:2]
```

```
'ab'
```

Та же логика будет работать и со списками, и с кортежами, и с NumPy-массивами, и с Pandas-датафреймами.

Как и с индексацией, параметры слайсинга могут быть отрицательными. Делая шаг −2, мы берём элементы с нулевого индекса по len(a) с шагом −2. Это означает последний, то есть минус первый элемент — букву d, затем, через два, минус 3-й (он же 2-й элемент с начала) — букву b.

```
# как и с индексацией в значениях m, n, k могуть быть и отрицательные целые числа
# с нулевого индекса по последний (включительно) с шагом -2
a[::-2]
```

```
'db'
```

Таким образом можно быстро и эффективно переворачивать строки и массивы данных. Частая запись — два двоеточия, минус 1.

Ссылка на Google Colab: https://drive.google.com/file/d/1ElPhd96BHtENIsXpcTOBlovbl0UKBHro/view?usp=sharing

```
[ ]  1  a[::-1]
```

Явное приведение типов

Как и в случае с цифрами, в строках можно вызвать явное приведение типов. Одноимённая с типом данных функция str приведёт разные переменные к типу str. Рассмотрим несколько примеров.

```
1  str(False)
```

```
'False'
```

```
[ ]  1  str(1.23)
```

```
'1.23'
```

```
[ ]  1  str(2)
```

```
'2'
```

```
1  str(1/3)
```

```
'0.3333333333333333'
```

```
[ ]  1  # пустая строка - ''
     2  str()
```

```
''
```

Функция len() принимает на вход строку или любую другую последовательность в Python и возвращает количество её элементов. В примере len(a) даёт нам 4 элемента.

```
[ ]   1   len(a) # 4 символа
```

```
4
```

Ключевое выражение in возвращает True, если подстрока полностью входит в строку. В примере bc входит в a целиком, поэтому мы получаем True.

```
[ ]   1   # вс
      2   'bc' in a
```

```
True
```

Встроенные методы

Расскажу немного о встроенных методах. Вы уже видели встроенные функции. Метод — такая же функция, неявно принимающая на вход значение переменной, из которой её вызвали. Для разных типов данных существуют разные встроенные функции. К ним можно получить доступ, поставив точку после переменной. Большинство таких функций возвращают некоторое значение.
Например, upper возвращает копию строки, все буквы которой приведены к верхнему регистру. Lower, наоборот, возвращает копию строки со всеми буквами в нижнем регистре.

		2. While В задачах на программирование нередко нужно повторять один и тот же код. Например, светофор должен последовательно выводить зелёный, жёлтый и красный цвета. Календарь должен обновляться каждый год, а форма ввода пароля — сбрасывать данные, пока вы не введёте нужную комбинацию. Можно копировать блоки кода и раз за разом вставлять их в программу, но рано или поздно они закончатся. Мы рассмотрим структуру, с помощью которой этого можно избежать, — циклы. При работе с моделями машинного обучения придётся обрабатывать большое число данных и выполнять однотипные действия. Это не проблема, если бы данных было немного, но речь идёт о десятках тысяч строк табличных данных. Не очень удобно для каждой строчки писать свою отдельную программу. Было бы неплохо иметь в своём арсенале структуру, которая сможет повторять заданные блоки кода нужное число раз. Поговорим о циклах. While На английском циклы называются loop, что переводится как «петля». Это хорошо объясняет их суть. Вернёмся к задаче с вводом пароля. В прошлых видео мы сделали проверку на корректность ввода. Но в случае, если пользователь вводил неправильный пароль, второй попытки у него не было. Как-то неправильно.	Ссылка Google Colab: https://colab.research.google.com/drive/19ySn1N1OX9_RWkFYQQ2h2XqUFo53-fUR?usp=sharing

```
password = input("Введите свой пароль: ")

if password == "qwerty123":
  print("Пароль правильный добро пожаловать")
else:
  print("Пароль неправиьлный, попробуй еще раз!")
```

Давайте исправляться. Нарисуем новую блок-схему. Теперь программа будет перезапускаться до тех пор, пока вы не введёте правильный пароль.

```
password = input("Введите свой пароль: ")

while password != "qwerty123":
  print("Пароль неправильный, попробуй еще раз!")
  password = input("Введите свой пароль: ")

print("Пароль правильный добро пожаловать")
```

Рассмотрим устройство цикла while. Как правило, цикл while используют, когда заранее неизвестно, сколько раз должно выполняться тело цикла. Зато известно условие, которого нужно дождаться. В нашем случае это правильный пароль.

Условие записывается сразу после ключевого выражения while. Цикл будет выполняться до тех пор, пока условие правдиво, то есть True. Код, записанный с отступом, принадлежит циклу. Как правило, его называют телом цикла.

```
[9]  print(10)
     print(9)
     print(8)
     print(7)
     print(6)
     print(5)
     print(4)
     print(3)
     print(2)
     print(1)
```

Рассмотрим другой пример. Представьте, что нам нужно вывести все числа от 10 до 1 включительно. Для решения «в лоб» мы бы записали следующий код.
Обратите внимание: строка print() всё время повторяется, меняется лишь её аргумент. Значит, такую запись можно упростить. Вместо числа мы возьмём переменную x, из которой будем вычитать единицу на каждом шаге. Вместо десяти функций print() будет всего одна.

Рассмотрим код x >= 1 — условие в примере. Оно следует после ключевого слова while. До тех пор, пока оно истинно, т.е. возвращает True, будет выполняться тело цикла. Переменная x будет появляться на экране и уменьшаться на единицу. На одиннадцатом запуске цикл прервётся, значение x станет равным 1, соответственно, x >= 1 выдаст нам False. Это говорит о том, что мы выходим из тела цикла и продолжаем итерироваться строка за строкой.

		Что будет, если изменить условие и убрать знак равно? Тогда вывод остановится на 2, ведь когда x станет единицей, условие цикла перестанет выполняться. Программу, где встречается цикл, удобно масштабировать. Изменим 10 на 100 и получим код, который считает от 100 до 0. Мы всего лишь изменили одну строчку. Синтаксис цикла while очень похож на условный оператор. Похож настолько, что существует расширенный цикл while, где есть else. По выходу из тела цикла он запустит соответствующий блок кода. Как правило, такую запись используют, когда цикл заканчивается извне, не дойдя до основного, но об этом позже. 3. For В противовес циклу while, for применяется в задачах, где заранее известно число итераций. Если вам нужно вывести 3 раза слово «Привет» или квадрат чисел, понадобится помощь for. Допустим, с сегодняшнего дня вы начали подтягиваться на турнике, и с каждым днём хотите делать на одно подтягивание больше. Рассмотрим синтаксис цикла for. После ключевого выражения for, через пробел, следует название переменной. В неё мы будем копировать значения из нашего диапазона. После переменной стоит in и название значения, по которому мы хотим пройтись. Обычно это список или некоторый диапазон.	

После, с отступом в 4 пробела или одну табуляцию, следует блок кода, который будет повторяться в цикле. Этот блок называется телом цикла.
Рассмотрим, какие значения хранятся в переменной i.

```
for i in range(365):
  print(i)
```

Она принимает значение от 0 до 364 включительно. Интересно, что после работы цикла переменной можно получить доступ. Храниться не будет последнее значение из диапазона.

```
[ ] print(i)

    364
```

За создание диапазона, по которому будет проходиться переменная, отвечает функция range(). Она создаёт диапазон значений в зависимости от аргументов:

- первый аргумент — начальное значение,
- второй аргумент — последнее значение, которое не включается в диапазон,
- третий аргумент — шаг.

Если указать только одно число, Python воспримет это как диапазон от нуля до этого значения с шагом один. Поэтому эти два выражения равны между собой.

```
for i in range(0,6,1):
  print(i)
```

```
0
1
2
3
4
5
```

```
[7] for i in range(6):
      print(i)
```

```
0
1
2
3
4
5
```

Break

Как прервать выполнение цикла? Для этого есть ключевое слово break. Представьте игру, в которой пользователю нужно угадать целое число за 5 попыток. В течение игры мы будем давать подсказки: больше это число или меньше предположения. Если человек угадает значение, игра закончится сразу. Мы остановим выполнение цикла с помощью break.

```
[10] print("Угадай число которое я загадал.")
     print("У тебя всего 5 попыток.")

     for i in range(5):
       a = int(input("Введи свое число: "))

       if a > 15:
         print("Меньше")
       elif a == 15:
         print("Угадал")
         break
       else:
         print("Больше")
```

```
Угадай число которое я загадал.
У тебя всего 5 попыток.
Введи свое число: 1
Больше
Введи свое число: 15
Угадал
```

То, что у нас появилась возможность досрочно закончить цикл, хорошо стыкуется с else, который запускается, если цикл отработал все свои шаги. Если мы дошли до else, значит, за 5 ходов человек так и не смог отгадать число и проиграл.

Continue

Что делать, если мы хотим пропустить текущую итерацию, но не прерывать выполнение цикла? Для этого существует оператор continue.
У нас есть строка, которую мы хотим побуквенно вывести на экран, но при этом, если она содержит точки, то их нужно пропустить. Для этого воспользуемся конструкцией continue.

Важно: конструкции break и continue работают и в цикле while.

Вывод

- Простой с виду строчный тип данных хранит в себе множество секретов. При детальном рассмотрении оказывается, что это вообще не строки, а хитрая комбинация цифр. Мы узнали, что строки можно «нарезать», как нам хочется, при помощи срезов. Познакомились со встроенными методами, которые

		служат самым разным целям, от понижения регистра до удаления лишних пробелов. • Циклы — уникальные структуры. Они позволяют перезапускать нужные участки кода. Мы познакомились с циклом while. Как правило, он используется, когда заранее неизвестно, сколько раз должно выполняться тело цикла. Зато известно условие, при выполнении которого должен выполняться этот цикл. • Мы познакомились и со второй разновидность циклов — for. Это отличный помощник, когда нам заранее известно число шагов или есть диапазон, по которому можно пройтись. Одна из таких структур — массивы. О них мы скоро поговорим.	
Закрепление изученного материала	25 мин.	Закрепление: операции над строками Задача 1 Дана переменная string, в которой нужно заменить все знаки препинания на смайлики (':)'). Знаками препинания считай символы: точка, запятая, двоеточие, тире (дефис), восклицательный и вопросительный знаки. Значение переменной string должно измениться в процессе выполнения программы. Для проверки используй фразу «Тяжёлая интернет-зависимость — это когда ты выходишь из интернета, а он из тебя нет».	

		Задача 2 Напиши программу, которая принимает на вход слово (word) и последовательно выводит все гласные из него. Для проверки используй word = 'Зимушка-зима'. Закрепление: цикл while в Python Задача 3 Ты играешь в компьютерную игру, дошёл до схватки с финальным боссом, но вот беда — компьютер «заглючил», и ты не можешь управлять персонажем в игре. Босс атакует и каждую секунду наносит один удар, который отнимает 80 единиц здоровья. Создай цикл, который позволяет понять, через сколько секунд босс победит, если на начало схватки у персонажа было 500 единиц здоровья. В результате работы программа должна вывести на экран количество секунд, в течение которых будет длиться схватка. Ответ должен быть выведен на экран в виде целого числа без какого-либо дополнительного поясняющего текста. Используй переменную current_health для сохранения текущего уровня здоровья, изменяя её по ходу цикла, и переменную attack = 80 для хранения значения атаки Босса. Закрепление: цикл for в Python Задача 4 Выведи таблицу квадратов и кубов для чисел от 1 до 10 включительно.	

		Каждая новая строчка раскрывается через print() и имеет формат 2 4 8, то есть число, затем его квадрат, а затем его куб.	
Этап подведения итогов занятия (рефлексия)	5 мин.	Вопросы для обсуждения • Что тебе более всего понравилось на уроке? • Что было трудным или непонятным?	Педагог способствует размышлению обучающихся над вопросами
Информация о домашнем задании, инструктаж по его применению	3 мин.	Домашнее задание представляет из себя решение нескольких задач по программированию. Программы проверяются в автоматическом режиме на платформе Stepik или платформе Академия искусственного интеллекта.	

Рекомендуемые ресурсы для дополнительного изучения:

1. ПИТОНТЬЮТОР. [Электронный ресурс] – Режим доступа: http://pythontutor.ru/.
2. Онлайн игра на программирование CodeCombat:. [Электронный ресурс] – Режим доступа: https://codecombat.com/.
3. Прямая ссылка на начало игры. [Электронный ресурс] – Режим доступа: https://codecombat.com/play.