ТЕХНОЛОГИЧЕСКАЯ КАРТА ЗАНЯТИЯ

Тема занятия: Условный оператор.

Аннотация к занятию: на данном уроке обучающиеся знакомятся с условным оператором if. В первой части урока они учатся использовать if при написании кода на Python. Во второй части знакомятся с полным условным оператором и каскадным условным оператором, решают задачи.

Цель занятия: формирование знаний об условном операторе if и решение задач на языке программирования Python.

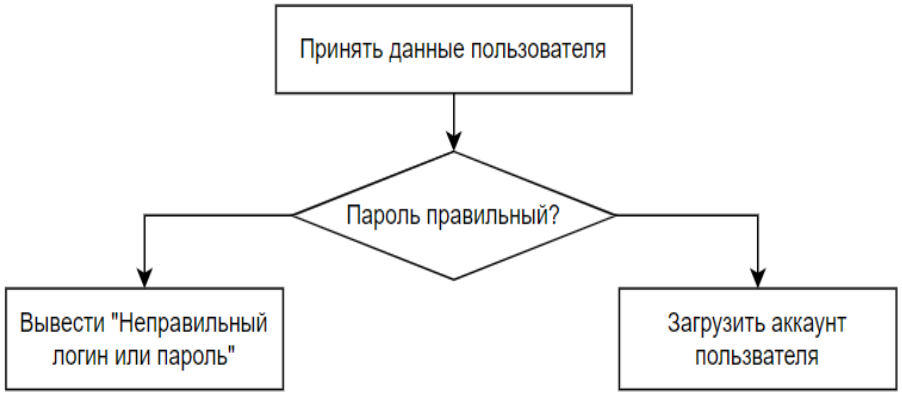
Задачи занятия:

- познакомить с краткой, полной и каскадной конструкциями условного оператора;
- научить применять условный оператор для решения разнообразных задач.

Ход занятия

Этап занятия	Время	Деятельность педагога	Комментарии, рекомендации для педагогов
Организационный этап	2 мин.	Добрый день! Как у вас настроение? Я рада вас видеть!	
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? **Возможные ответы 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

		Озвучивается план, в соответствии с которым нужно разобраться, как работает условный оператор в Python. **Порядок тем для изучения:** ● Условный оператор if ● Конструкция if-else ● Полный условный оператор	

Изучение нового материала	50 мин.	Условный оператор
		Структура условного оператора:
		if логическое условие:
		результат, если условие выполняется
		 
		
		В Python проверка условия осуществляется при помощи оператора if («если»). Рассмотрим код для проверки пароля:
		`password = input("Введите пароль: ")`
		`if password == "qwerty123":`
		`print("Пароль правильный, добро пожаловать")`
		Первым в строке идёт ключевое слово if, далее — операция сравнения введённого пароля с тем, что хранится в базе, и двоеточие. Двоеточие сообщает Python, что со следующей строки пойдут команды, которые будут выполняться только в том случае,
			Ссылка на Google Colab:
			https://colab.research.google.com/drive/1z6eWU3XV8COfG8i7ZUMcTMOkTAFOG2bR?usp=sharing

если результат сравнения возвращает значение True. Строки, относящиеся к if, записываются с отступом в 4 пробела с новой строки.

```
password = input("Введите пароль: ")  ← Выполняется всегда

if password == "qwerty123":  ← Выполняется всегда
    print("Пароль правильный, добро пожаловать")  ←
                                Выполняется только если if True:
print("Хорошая погда, не так ли?")  ← Выполняется всегда
```

Запустим программу и посмотрим на её консольный вывод:

```
Введите пароль: qwerty123
Пароль правильный, добро пожаловать
Хорошая погда, не так ли?
```

```
Введите пароль: 123qqwe
Хорошая погда, не так ли?
```

Зафиксируем структуру ветвления:

if логическое условие:
    результат, если условие выполняется

Отступы — обязательный синтаксический инструмент при работе на языке Python. В языке Си для указания принадлежности кода к условию его помещают в кавычки.

```c
int main()
{
    printf( "Сколько вам лет? " );
    scanf( "%d", &age );

    if ( age < 18 ) {
        printf ("Вы родились после 2003 года \n" );
        printf ("Веротяно вы еще учитесь в школе\n" );
        printf ("Зато вы молоды душой и духом!\n" );
    }

}
```

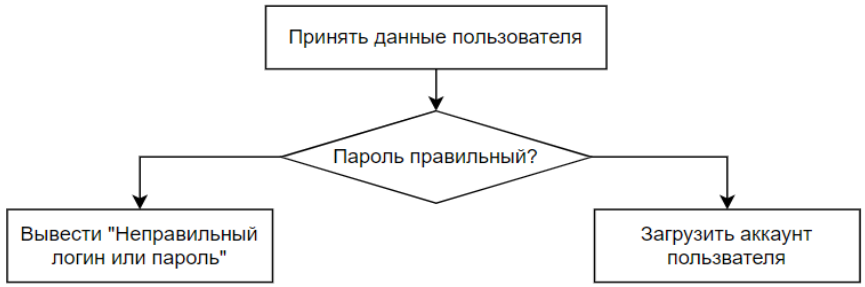
В Python вместо кавычек используют отступы

```python
age = int(input("Сколько вам лет?: "))

if age < 18:
    print("Вы родились после 2003 года")
    print("Веротяно вы еще учитесь в школе")
    print("Зато вы молоды душой и духом!")
```

Полный условный оператор

Вновь рассмотрим пример с проверкой пароля. Мы научили программу реагировать на правильный пароль, но как описать реакцию на некорректный ввод?



Понадобится полный логический оператор. Помимо if, он включает в себя ключевое слово else.

Else — полная противоположность if, блок кода, который выполняется только в том случае, если результат сравнения — False.

Программа будет выглядеть так:

```
if password == "qwerty123":
    print("Пароль правильный, добро пожаловать")
else:
    print("Пароль неправильный, попробуйте еще раз")
```

В новой программе обрабатываются все случаи:

- когда пользователь ввёл правильный пароль, выражение возвращает True и на экран выводится сообщение «Пароль правильный»;
- когда пароль не совпадает с верным, выражение возвращает False, а на экран выводится сообщение «Пароль неправильный».

Логические операции

Одновременно в if может быть записано только одно условие, но за счёт логических операций можно сделать из пары простых условий одно сложное. В этом помогут логические операторы: они выступят в качестве «клея» между выражениями:

- and — логическое И — связывает два условия и возвращает True только в случае, если оба условия верные;
- or — логическое ИЛИ — связывает два условия и возвращает True тогда, когда хотя бы один операнд равен True;
- not — логическое НЕ (отрицание) — применяется к одному условию. Логическое НЕ возвращает True, если операнд равен False, и наоборот.

Логическое И
В случае с Петей, для его выхода из дома должно выполняться сразу два события. Воспользуемся оператором and.

```
weather = input("Какая погода на улице? ")
friends = input("Где находяться друзья? ")

if weather == "Отличная" and friends == "На улице":
    print("Петя выйдет")
else:
    print("Петя будет сидеть дома")
```

За счёт оператора and мы соединили два коротких выражения и получили одно большое.

		Новое выражением станет истинным, только если каждое из составных выражений истинно. Иными словами, and означает «и то, и другое». Где ещё это может быть полезно? Например, если нужно проверить совпадение числа с диапазоном. Пусть на сайте нужно определить, является ли посетитель школьником, на основе его возраста. Важно отметить, что логические операторы имеют наименьший приоритет и выполняются после всех арифметических операций и сравнений. Попробуем задать программе число 10. Оно попадает в нужный диапазон и удовлетворяет обоим выражениям. Результат оператора and в таком случае — True. А вот если задать число 20, то операторы сравнения выдадут отличающийся друг от друга результат. Результат and будет False. Оператор and может объединять любое количество выражений. За счёт этого можно генерировать условия любой длины. Перед вами проверка того, наступил ли Новый год:	Ссылка на Google Colab: https://colab.research.google.com/drive/16LTslMuQg0T6pzbX_edAV-02mHDvOBUC?usp=sharing

```
mounth = int(input())
day = int(input())
hours = int(input())
minutes = int(input())
seconds = int(input())

if mounth == 12 and day == 31 and hours == 23 and minutes == 59 and seconds == 59:
    print("С новым годом!")
else:
    print("Еще рано!")
```

Оператор and стоит сравнить с умножением:

- если первое утверждение истинно (True), а второе ложно (False), то результат 1 * 0 = 0 False;
- если оба ложны (False), то результат 0 * 0 = 0 ложь (False).
- если оба верны (True), то результат 1 * 1 = 1 истина (True).

Логическое ИЛИ
Для возвращения True в программу требуется, чтобы хотя бы одно из условий выполнялось. Например, если бы Петя из прошлого примера был менее требовательным и для его выхода из дома требовалось хотя бы одно условие, программа имела бы вид:

```
weather = input("Какая погода на улице? ")
friends = input("Где находяться друзья? ")

if weather == "Отличная" or friends == "На улице":
    print("Петя выйдет")
else:
    print("Петя будет сидеть дома")
```

Если заменить логический оператор на or, то новое выражение будет истинным, если хотя бы один из операндов является истинным. Иначе выражение ложное.

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

OR можно сравнить со сложением, где нельзя превысить максимальное значение — 1. Рассмотрим возможные случаи.

- Если первое утверждение истинно, а второе ложно, то результат 1 + 0 = 1 (истина).
- Если же, наоборот, первое ложно, а второе истинно, то результат 0 + 1 = 1 (истина).
- Если оба верны, то результат 1 + 1 = 1 (не забываем про максимальное значение), то есть снова истина.
- Если оба неверны, то результат 0 + 0 = 0 (ложь).

Логическое НЕ
Логическое отрицание или логическое не, в отличие от прошлых логических операторов, применяется только к одному элементу. Оно меняет условие на противоположное:
True = not False

		После того, как друзья Пети составили алгоритм его поведения, они поняли, что программа нуждается в доработке. Петя выходит на улицу только в случае, если на улице не идёт дождь.
		`weather = input("Какая погода на улице? ")`
		`#friends = input("Где находяться друзья? ")`
		
		`if not weather == "Дождь":`
		`  print("Петя выйдет")`
		`else:`
		`  print("Петя будет сидеть дома")`
		
		Где оператор not может быть полезен? Например, если нам известно, что некоторый диапазон чисел не должен участвовать в выборке:
		`age = int(input("Сколько вам лет?"))`
		
		`if not(18 > age):`
		`  print("Не хотите поучаствовать в нашей акции?")`
		
		Вывод
		Логические операторы могут обработать только одно выражение. Если есть необходимость в запуске кода при выполнении нескольких условий, их можно объединить за счет логических операторов. Согласитесь, они чем-то напоминают знакомые нам алгебраические операторы, которые мы видели ранее.
			Ссылка на Google Colab:
			https://colab.research.google.com/drive/1RF6zw9ikomVD-ngcKjwikSuJ6Gdq5y9Z?usp=sharing

Логическое И (and)

a	b	a and b
False	False	False
False	True	False
True	False	False
True	True	True

Оператор and можно сравнить с умножением:

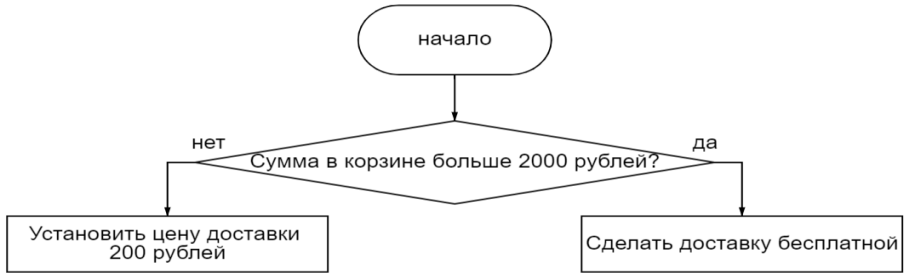
- Если первое утверждение истинно (True), а второе ложно (False), то результат 1 * 0 = 0 False;
- Если оба ложны (False), то результат 0 * 0 = 0 ложь (False).
- Если оба верны (True), то результат 1 * 1 = 1 истина (True).

Логическое Или (or)

a	b	a or b
False	False	False
False	True	True
True	False	True
True	True	True

OR можно сравнить со сложением, где нельзя превысить максимальное значение — 1. Рассмотрим возможные случаи.

		• Если первое утверждение истинно, а второе ложно, то результат 1 + 0 = 1 (истина). • Если же, наоборот, первое ложно, а второе истинно, то результат 0 + 1 = 1 (истина). • Если оба верны, то результат 1 + 1 = 1 (не забываем про максимальное значение), то есть снова истина. • Если оба неверны, то результат 0 + 0 = 0 (ложь). Логическое НЕ (not) Логическое отрицание или логическое не, в отличие от прошлых логических операторов применяется только к одному элементу. Оно меняет условие на противоположное: True = not False Каскадный условный оператор (плохой пример) В своей полной форме условный оператор может обработать два действия: • если событие произошло, • если событие не произошло. Что делать, если нужно, чтобы программа генерировала несколько вариантов ответа? Всё просто: если для двух событий достаточно одного условного оператора, то для обработки трёх понадобится добавить ещё один оператор. Рассмотрим такой пример. В приложении по доставке еды действовала акция на бесплатную доставку при заказе от 2 000 рублей. Это легко отобразить на схеме:	



С притоком новых пользователей акция изменилась и стала ступенчатой. Теперь при заказе:

- до 1000 рублей доставка составит 200 рублей,
- от 1000 до 2000 рублей доставка составит 100 рублей,
- от 2000 рублей доставка будет бесплатной.

Новая схема алгоритма выглядит так:

Если изначально акция выглядела так:

```
if price_amount > 2000:
    delivery_amount = 0
else:
    delivery_amount = 200
```

То для добавления в программу новой «ветви», пришлось добавить условие:

```
if price_amount > 2000:
    delivery_amount = 0
else:
    if price_amount > 100:
        delivery_amount = 100
    else:
        delivery_amount = 200
```

По аналогии с функциями, условия, помещенные внутрь других условий, называются вложенными. Внутри условного оператора можно использовать любые функции или конструкции языка Python, которые вам известны, без опасения, что компилятор выдаст ошибку.

Именно поэтому в коде возможны такие формы записи:

```
if условие1:
    блок кода
else:
    if условие2:
        блок кода
    else:
        if условие3:
            блок кода
        ...
```

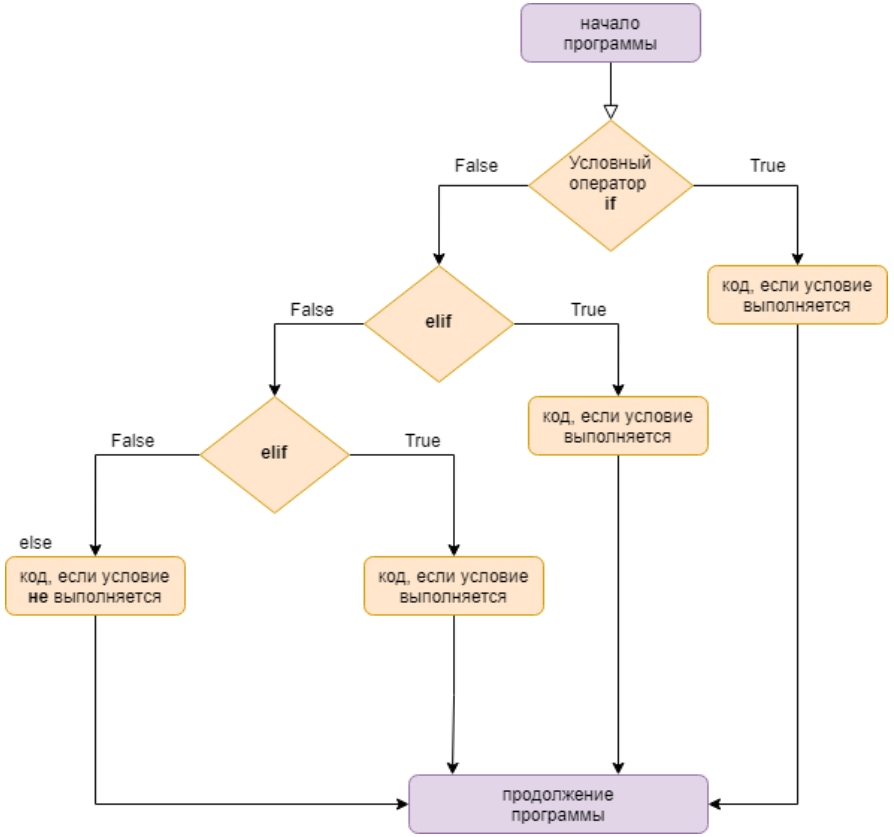
```
if условие :
    блок кода
else:
    if условие :
        блок кода
    else:
        блок кода
    if условие :
        блок кода
    else:
        блок кода
    if условие :
        блок кода
    else:
        блок кода
```

```
a = int(input())
b = int(input())
operation = input()

if operation == "+":
    print(a+b)
else:
    if operation == "-":
        print(a-b)
    else:
        if operation == "*":
            print(a*b)
        else:
            if operation == "/":
                if b == 0:
                    print("На ноль делить нельзя!")
                else:
                    print(a/b)
            else:
                print("Неверная операция")
```

		Умение базовых конструкций языка быть вложенными — понятный принцип, который позволяет создавать объёмные программы. Несовершенство условного оператора можно перекрыть вложенностью.
		
		Каскадный условный оператор (хороший пример)
		
		Python как продвинутый язык предлагает несколько решений одной и той же проблемы разной степени элегантности. Код прошлой задачи может вызвать у тебя такие мысли как «а не слишком ли он объемный?», «можно ли решить эту задачу быстрее?», «кажется, создатели курса хотят подвести меня к новой теории» и т.д. Всё это — правильные мысли! Мы рассмотрим связанные условия или каскадный условный оператор, который иногда называют цепным условным оператором.
		
		Структура условного оператора:
		if первое логическое условие:
		  результат, если первое условие выполняется
		elif второе логическое условие:
		  результат, если второе условие выполняется
		elif третье логическое условие:
		  результат, если третье условие выполняется
		...
		else:
		  результат, если ни одно условие не выполняется
		
		В отличие от if и else, elif может повторяться множество раз, если необходимо проверить много условий.	

		Каскадный условный оператор упрощает работу, когда в Python нужно проверить несколько условий. Код прошлой задачи превращается в компактную запись:
		`a = int(input())`
		`b = int(input())`
		`operation = input()`
		
		`if operation == "+":`
		`  print(a+b)`
		`elif operation == "-":`
		`  print(a-b)`
		`elif operation == "*":`
		`  print(a*b)`
		`elif operation == "/":`
		`  if b == 0:`
		`  print("На ноль делить нельзя!")`
		`  else:`
		`  print(a/b)`
		`else:`
		`  print("Неверная операция")`
		
		Интересно, что операторов elif в такой записи может быть неограниченное число. Кроме того, у этого оператора есть особенность, которая перешла от вложенной формы. Посмотрим на её блок-схему:	



При проверке такого условного оператора сначала проверяется первое условие, записанное в if. Если оно истинно, то выполняется его блок кода, а остальная часть оператора игнорируется. Однако, если условие ложно, оператор переключается на первый elif и проводит аналогичную проверку.

```
if operation == "+":
    print(a+b)
elif operation == "-":
    print(a-b)
elif operation == "*":
    print(a*b)
elif operation == "/":
```

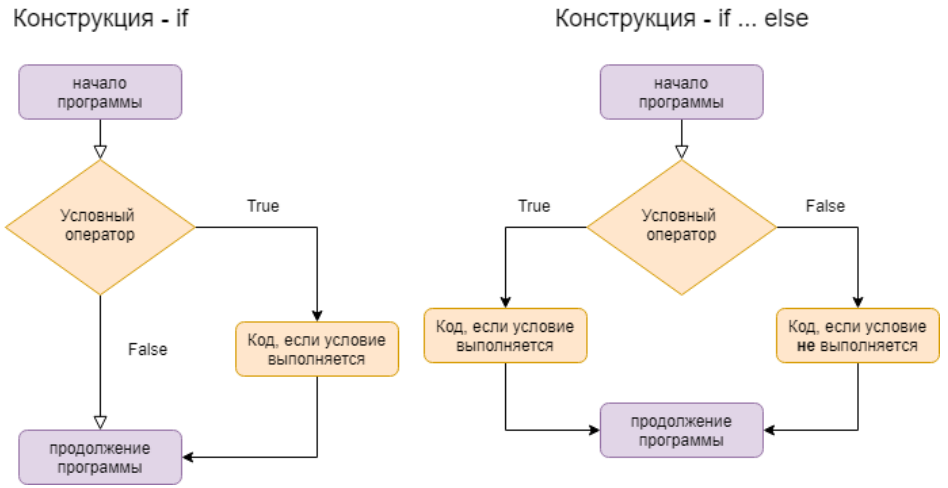
Если в такой конструкции одно из условий сработало, то все остальные отбрасываются, что экономит время работы программы. На основе этого правила можно сделать два вывода.

1. В первые строчки оператора имеет смысл ставить наиболее популярные события (если это возможно).
2. Каскадным условным оператором лучше проверять значения одной переменной или пары значений.

Вывод

1. Важно отметить, что одновременное срабатывание одной пары if и else невозможно. Вспомните принципы устройства выражения: оно может быть или ложным, или истинным, промежуточного результата быть не должно. Перенесём это на ввод пароля: нет такого события, в котором пользователь бы ввёл правильный и неправильный одновременно.

Рассмотрим блок-схему if и if-else. Как видите, у полного логического оператора на одну ветвь больше за счёт добавления в конструкцию else.

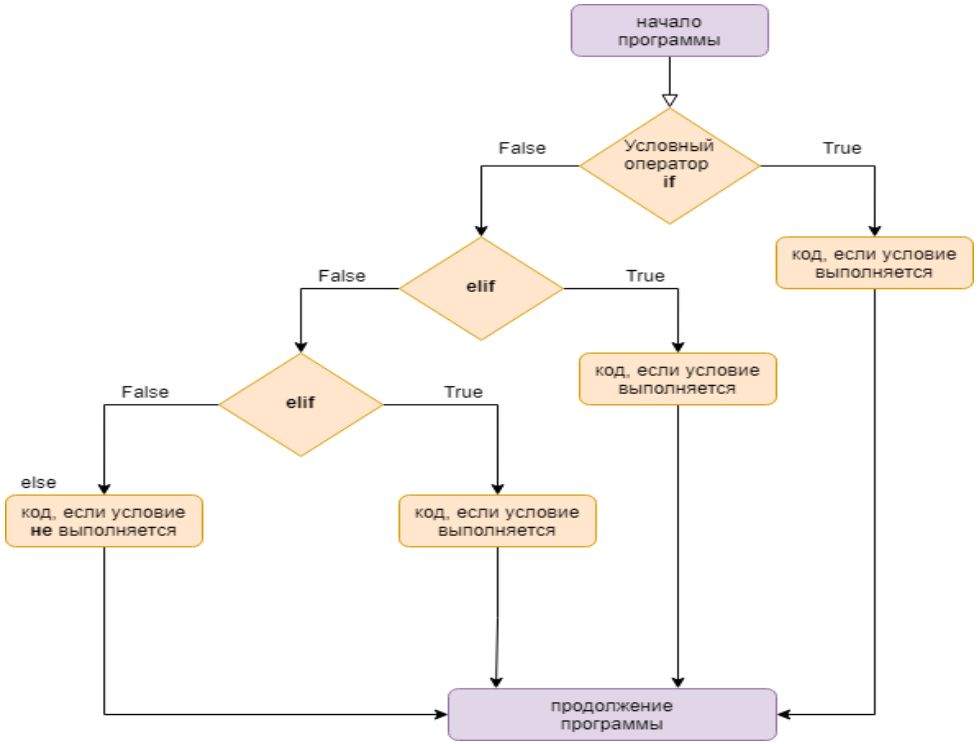


Закрепим структуру условного оператора с блоком else:
if логическое условие:
результат, если условие выполняется else:
результат, если условие не выполняется

Важно! Оператор else тоже требует после себя двоеточия и отступа с новой строки.

2. Каскадный условный оператор затрагивает важную философию программирования. Задачу, на которую он направлен, можно решить с помощью классического условного оператора,

		вот только это будет выглядеть громоздко. Помните: если ваше решение кажется гигантским, а строки кода однотипны и повторяются, скорее всего, его можно упростить.
		
		Структура каскадного условного оператора:
		if первое логическое условие:
		  результат, если первое условие выполняется
		elif второе логическое условие:
		  результат, если второе условие выполняется
		elif третье логическое условие:
		  результат, если третье условие выполняется
		...
		else:
		  результат, если ни одно условие не выполняется
		
		В отличие от if и else, elif может повторяться множество раз, если необходимо проверить много условий.
		
		Блок-схема цикла if-elif-else выглядит так:	


начало программы
Условный оператор if
False
True
код, если условие выполняется
elif
False
True
код, если условие выполняется
elif
False
True
else
код, если условие не выполняется
код, если условие выполняется
продолжение программы

Закрепление изученного материала	15 мин.	**Решается задача из блокнота.** Индекс массы тела (ИМТ) является простым и одновременно важным индикатором состояния здоровья человека. Он рассчитывается как отношение веса человека (в килограммах) к квадрату его роста (в метрах). Если ИМТ находится в пределах от 18,5 до 24,99 включительно, соответствие между массой тела и ростом человека считают нормальным. Значения ниже этого диапазона сигнализируют о недостаточной массе тела, выше — об избыточной массе тела. Помогите врачам автоматизировать принятие решений на основе ИМТ. Напишите программу, в которой человек может указать свой вес (weight) и рост (height), а затем прочитать заключение: «Недостаточная масса тела», «Норма», «Избыточная масса тела». Для проверки используй следующие показатели: вес = 85 рост = 1,83	
Этап подведения итогов занятия (рефлексия)	8 мин.	**Вопросы для обсуждения** • Что вы открыли для себя на этом уроке? • С чем были сложности?	Педагог способствует размышлению обучающихся над вопросами
Информация о домашнем задании, инструктаж по его применению	5 мин.	Домашнее задание представляет из себя решение нескольких задач по программированию. Программы проверяются в автоматическом режиме на платформе Stepik или платформе Академии 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.