СОДЕРЖАНИЕ

Условный оператор

До этого момента все программы, которые мы писали, были простыми и линейными: они получали данные, выполняли какие-то вычисления и выводили результат на экран. Например, мы решали задачи, как сложить два числа, вывести их сумму, или найти цифры числа. Однако в реальной жизни часто нужно учитывать различные условия. Например, перед покупкой билета на поезд система должна проверить, достаточно ли средств на карте, или если студент вводит свой возраст, система может проверить, корректно ли он ввёл данные.

Для того чтобы программа могла принимать решения на основе определённых условий, в Python есть специальный инструмент — условный оператор.

Условный оператор — это конструкция, которая позволяет выполнять тот или иной код в зависимости от выполнения заданного условия. Основной оператор для создания таких условий в Python — это оператор **if**. Синтаксис оператора if:

```
if условие:
    блок_кода
```

Здесь **условие** — это выражение, которое должно быть истинным или ложным. Если условие истинно (True), то выполняется **блок кода**, иначе этот блок пропускается. Например:

```
age = 18
if age >= 18:
    print("Вам можно голосовать")
```

В этом примере проверяется, больше или равно ли значение переменной age числу 18. Если условие выполняется, программа выводит сообщение на экран.

Булевый тип данных

Чтобы понять, как работают условия, нужно познакомиться с **булевым типом данных** (или **логическим типом**). Булевое значение может быть только двух видов:

- True (истина)
- False (ложь)

Когда мы проверяем условие в операторе if, результат этой проверки всегда будет булевым значением — либо True, либо False.

Операторы сравнения

Для создания условий используются операторы сравнения. Они сравнивают два значения и возвращают True или False.

Оператор	Произношение	Пример	Результат
==	Равно	5 == 5	True
!=	Не равно	5 != 4	True
>	Больше	5 > 3	True
<	Меньше	3 < 5	True
>=	Больше или равно	5 >= 5	True
<=	Меньше или равно	4 <= 5	True

Важно: операторы !=, >=, <= всегда пишутся без пробелов между знаками (например, !=, а не ! =).

Отступы и блок кода

В Python структура программы определяется **отступами**. Блок кода, который выполняется при выполнении условия, должен находиться на **4 пробела** правее, чем строка с оператором if. Это важно! Если вы забудете поставить отступ, Python выдаст ошибку.

```
age = 20
if age >= 18:
    print("Вы совершеннолетний")
    print("Добро пожаловать!")
```

Оба print() находятся в блоке if и выполняются только если условие истинно.

Оператор else

Иногда нужно выполнить один блок кода, если условие истинно, и другой, если оно ложно. Для этого существует оператор else. Он идёт сразу после if и выполняет свой блок кода, если условие в if не выполнено.

```
age = 16
if age >= 18:
    print("Вы совершеннолетний")
else:
    print("Вы ещё не достигли 18 лет")
```

Оператор elif

Если нужно проверить несколько условий, используется оператор elif (сокращение от "else if"). Он позволяет добавить дополнительные проверки после основной.

```
age = 16
if age >= 18:
    print("Вы совершеннолетний")
elif age >= 16:
    print("Вы можете водить")
else:
    print("Вы ещё молоды для важных решений")
```

Здесь программа проверяет несколько условий последовательно. Сначала она проверяет, достиг ли возраст 18 лет, если нет — проверяет, равен ли он 16 или больше, и, наконец, если ни одно из условий не выполнено, программа выводит сообщение из блока else.

Важно: в одной конструкции if-elif-else может быть сколько угодно блоков elif, но блок else всегда может быть только один, и он идёт последним.

Ошибки:

1) Нельзя писать elif, не указав условие:

```
if x > 5:
    print("больше 5")
elif:  # Ошибка!
    print("ошибка")
```

2) Нельзя писать условие после else:

```
if x > 5:
    print("больше 5")
else x < 5:  # Ошибка!
    print("меньше 5")
```

Логические операторы

Чтобы проверить сразу несколько условий, можно использовать логические операторы.

Оператор	Название	Пример использования	Результат
and	И	x > 5 and x < 10	True, если оба условия истинны
or	Или	x < 5 or x > 10	True, если хотя бы одно условие истинно
not	Не	not(x > 5)	Возвращает противоположное значение

Пример:

```
x = 7
if x > 5 and x < 10:
    print("Число больше 5 и меньше 10")
```

Таблица истинности для логических операторов

A	B	A and B	A or B	not A
True	True	True	True	False
True	False	False	True	False
False	True	False	True	True
False	False	False	False	True

Приоритет логических операторов:

1) not

2) and

3) or

Чтобы изменить приоритет выполнения, используйте скобки. Например:

```
x = 5
if (x > 2 or x < 1) and x < 10:
    print("Это число в нужном диапазоне")
```

Сравнение строк в Python

Когда речь идет о строках, важно понимать, что сравнение происходит не так, как мы привыкли в повседневной жизни. Python сравнивает строки на основе кодов символов в таблице Unicode. Этот процесс часто может удивить начинающих, ведь программа видит строки иначе, чем человек.

Когда программа сравнивает две строки, она начинает с первого символа каждой строки и проверяет их. Если они одинаковы, программа переходит к следующему символу и так далее, пока не найдет различие или не закончит сравнение всех символов.

```
s1 = "apple"
s2 = "apples"
print(s1 < s2)  # True
```

Здесь строка "apple" короче, чем "apples", но все символы в ней совпадают. Для программы это значит, что "apple" "меньше" по длине, и она считается меньшей строкой.

В таблице Unicode заглавные буквы имеют меньшие коды, чем строчные. Это значит, что:

```
print("A" < "a")  # True
```

Заглавная буква "A" "меньше", чем строчная "a". Это важно учитывать при работе со строками, так как это может привести к неожиданным результатам при сортировке строк.

Цифры также имеют свои коды в Unicode. При сравнении строки, содержащей цифры, с строкой, содержащей буквы, программа также опирается на их порядковое значение в таблице Unicode. Цифры идут раньше букв:

```
print("1" < "a")  # True
```

Здесь "1" меньше, чем "a", потому что символы с кодами, представляющими цифры, находятся раньше в таблице Unicode.

Если одна строка полностью содержится в другой, программа сравнивает длину этих строк.

```
print("cat" < "catalog")  # True
```

Для программы "cat" меньше, чем "catalog", потому что вторая строка длиннее, хотя они начинаются одинаково.

Полезные советы для сравнения строк:

1) Если требуется избежать различий между заглавными и строчными буквами, строки можно привести к одному регистру перед сравнением:

```
print("Apple".lower() == "apple".lower())  # True
```

2) Для точного сравнения строк нужно учитывать их длину и содержимое каждого символа, помня о правилах Unicode.

Сравнение разных типов данных в условных операторах

В Python сравнение разных типов данных не всегда возможно и не всегда правильно с точки зрения логики программы. Например, попытка сравнить строку с числом приведёт к ошибке. Это связано с тем, что такие типы данных, как строки и числа, интерпретируются по-разному, и Python не может автоматически определить, как их сравнить.

```
print("10" > 5)  # Ошибка TypeError
```

Здесь программа не понимает, как сравнить строку "10" с числом 5, поэтому возникнет ошибка TypeError. Сравнивать строки с числами в Python нельзя.

Однако, числовые типы данных int и float можно сравнивать между собой, потому что оба относятся к числам и Python автоматически приводит их к общему типу для сравнения.

```
print(10 > 5.5)  # True
```

Оператор in

Оператор in используется для проверки наличия элемента в последовательности (строке, списке, множестве и т.д.). Это простой и эффективный способ выяснить, содержится ли определённый символ или подстрока в строке.

```
word = "Python"
if "y" in word:
    print("Буква 'y' найдена в строке 'Python'!")
else:
    print("Буква 'y' не найдена.")
```

Этот код проверяет, содержится ли символ 'y' в строке "Python". Если символ найден, то программа выполнит первую ветвь с print().

Оператор in также можно использовать для проверки наличия подстроки в строке:

```
text = "Programming in Python is fun"
if "Python" in text:
    print("В тексте содержится слово 'Python'")
```

Здесь программа проверяет, содержится ли слово "Python" в переменной text. Оператор in удобен, когда нужно искать отдельные символы или части строк, а также подходит для работы с другими последовательностями, такими как списки.