IT CAREER HUB

Урок 5.

Операторы ветвления

IT CAREER HUB

Операторы ветвления

Операторы ветвления (или условные операторы) — это конструкции, которые позволяют выполнять различные блоки кода в зависимости от выполнения условий. С их помощью программа может принимать решения: если условие истинно, выполняется один блок кода, если ложно — другой.

В Python основными операторами ветвления являются:

- `if` — проверяет условие, и если оно истинно, выполняет блок кода.
- `elif` (сокращение от "`else if`") — проверяет дополнительное условие, если предыдущие были ложными, выполняет блок кода если истинно.
- `else` — выполняет блок кода, если все предыдущие условия оказались ложными.

Эти операторы также являются ключевыми словами.

Ключевые слова — это зарезервированные слова, которые имеют специальное значение и используются для выполнения определённых операций или действий.

Ключевые слова не могут быть использованы в качестве имён переменных или других идентификаторов, так как их функционал строго определен. Примеры ключевых слов: `if`, `else`, `while`, `for`, `def`, `return`, `import` и т.д.

Каждое ключевое слово выполняет конкретную задачу и определяет структуру программы, её логику и поведение.

По ссылке ниже можно найти полный список ключевых слов:

Официальная документация по ключевым словам Python

IT CAREER HUB

Оператор if

Оператор if — это один из ключевых операторов ветвления в Python, который позволяет программе выполнять определённый блок кода (тело), если заданное условие является истинным (True). Он используется для принятия решений и создания логики в коде.

Синтаксис оператора `if`:

```python
if условие:

    # начало блока кода или тела

    # код, который выполнится, если условие истинно

    # код может занимать любое количество строк, пока не закончится отступ

    # конец блока кода или тела

# код на этой строке не является частью блока if
```

- Условие — это выражение, которое может быть истинным (`True`) или ложным (`False`).
- Если условие истинно, выполняется блок кода под оператором `if`. Если ложно — программа пропускает этот блок.

Пример

```python
x = 10
```

```
if x > 5:
    print("x больше 5")       # выполнится
y = 1
if y == 0:
    print("y равняется 1")    # не выполнится
```

Особенности:

1. Блок кода под `if` должен иметь правильный отступ (4 пробела или один таб).

2. Если условие ложно, блок кода под **`if`** пропускается.

Неявное преобразование в bool

Неявное преобразование в `bool` происходит в конструкциях, где ожидается логический тип данных. К примеру это происходит в контексте операторов ветвления.

Когда в операторе `if` используется не булевое значение, оно автоматически преобразуется в логическое значение.

Пример

1. **True**:

```python
a = 5

if a:  # Неявное преобразование числа 5 из переменной a в True

print("Число 5 интерпретируется как True")
```

2. **False:**

```python
s = ""

if s:  # Неявное преобразование пустой строки в False

    print("Пустая строка интерпретируется как False")
```

IT CAREER HUB

Оператор elif

Оператор elif (сокращение от "else if") используется для проверки дополнительных условий в конструкции ветвления после оператора if. Он позволяет добавить альтернативные условия, которые проверяются только в случае, если предыдущее условие оператора if или другого elif оказалось ложным.

Синтаксис оператора `elif`:

```python
if условие1: # Конструкция обязательно должна начинаться с оператора if

    # Выполняется, если условие1 истинно.

elif условие2:

    # Выполняется, если условие1 ложно, а условие2 истинно

elif условие3:

    # Выполняется, если предыдущие условия ложны, а условие2 истинно

...
```

Важно:

- Можно использовать несколько блоков `elif` для проверки нескольких условий.
- Оператор `elif` не может быть использован без оператора `if`.
- Может быть выполнено только одно или ни одного условия.

IT CAREER HUB

Пример

Python

```python
x = 7

if x > 10:

    print("x больше 10")

elif x > 5:

    print("x больше 5, но меньше или равно 10")

age = 25

if age < 18:

   print("Несовершеннолетний")

elif age < 30:

   print("Молодой взрослый")

elif age < 50:

   print("Взрослый")
```

IT CAREER HUB

Оператор else

Оператор else используется вместе с операторами if и elif для выполнения кода, если предыдущие условия были ложными. Он помогает обрабатывать случаи, когда основные условия не выполняются, и задаёт альтернативный сценарий выполнения программы.

Синтаксис оператора `else`:

```python
if условие: # Конструкция обязательно должна начинаться с оператора if

    # Выполняется, если условие истинно (True)

elif условие2: # Оператор elif не обязателен

    # Выполняется, если условие1 ложно, а условие2 истинно

else:

    # Выполняется, если условие ложно (False)
```

Пример

```python
x = 3

if 1 < x < 5:

    print("x между 1 и 5")

else:

    print("x не между 1 и 5")
```

```
age = 25

if age < 18:

   print("Несовершеннолетний")

elif age < 30:

   print("Молодой взрослый")

elif age < 50:

   print("Взрослый")

else:

   print("Пожилой")
```

Особенности оператора `else`:

- Если все условия в блоках `if` и `elif` оказались ложными, автоматически выполняется блок кода, указанный в `else`.
- В отличие от `if` и `elif`, оператор `else` не требует никаких условий. Он просто выполняется, если предыдущие условия были ложными.

IT CAREER HUB

Задание для закрепления 1

1. Какой результат выведет следующий код?

```python
x = ""

if x:

    print("Внутри if")

else:

    print("Внутри else")
```

a. Внутри `if`
b. Внутри `else`
c. Ошибка
d. Программа ничего не выведет

Посмотреть ответ

2. Какой результат выведет следующий код? Выберите все правильные варианты.

```python
x = 25

if x > 10:

    print("x больше 10")

if x > 20:

    print("x больше 20")

else:
```

```
    print("x меньше или равно 20")
```

a. x больше 10
b. x больше 20
c. x меньше или равно 20
d. Ошибка

Посмотреть ответ

Использование нескольких операторов if

Иногда в программе нужно проверять несколько условий, которые не связаны между собой. Для этого можно использовать **несколько независимых операторов** `if`, каждый из которых будет проверять своё условие и выполнять соответствующий блок кода, если условие истинно.

Синтаксис с несколькими if:

```python
if условие1:

    # Выполняется, если условие1 истинно



if условие2:

    # Выполняется, если условие2 истинно



if условие3:

    # Выполняется, если условие3 истинно
```

В данном случае каждое условие проверяется отдельно, независимо от других условий. Это означает, что все операторы `if` могут выполняться одновременно, если все условия истинны.

Пример использования нескольких if:

```python
x = 7
```

```
if x > 5:
    print("x больше 5")  # Это условие истинно, выполнится

if x < 10:
    print("x меньше 10")  # Это условие тоже истинно, выполнится

if x == 7:
    print("x равно 7")  # Это условие также истинно, выполнится
```

Важно:

- В отличие от конструкции с `if-elif-else`, несколько операторов `if` не останавливают выполнение последующих проверок. То есть, если одно условие истинно, это не исключает выполнение других операторов `if`.
- Если необходимо, чтобы выполнилось только одно из условий, следует использовать `if-elif-else`.

Вложенные условия

Вложенные условия — это конструкции, когда один оператор if находится внутри другого оператора if. Это позволяет проверять дополнительные условия, если первоначальное условие оказалось истинным. Вложенные условия полезны, когда выполнение определённого блока кода зависит от выполнения нескольких условий.

Синтаксис вложенных условий:

```python
if условие1:
    if условие2:
        # код, который выполнится, если оба условия истинны
    else:
        # код, который выполнится, если условие1 истинно, а условие2 ложно
else:
    # код, который выполнится, если условие1 ложно
```

Важно:

- Можно использовать любое количество вложенных условий.
- Каждый уровень вложенности нужно сдвигать на новый уровень отступа.

Примеры

1. Проверка прав пользователя

Представьте, что нужно проверить, является ли пользователь администратором, а затем уточнить, имеет ли он доступ к определённым разделам.

```python
role = "admin"

section_access = True



if role == "admin":

    print("Вы администратор.")

    if section_access:

        print("У вас есть доступ к разделу.")

    else:

        print("У вас нет доступа к разделу.")

else:

    print("Вы не администратор.")
```

2. Определение условий для отпуска

Система проверяет, запланирован ли отпуск сотрудника, и затем уточняет, достаточно ли у него доступных дней для отпуска.

```python
vacation_requested = True

available_days = 5

required_days = 7
```

```
if vacation_requested:
    print("Запрос на отпуск получен.")
    if available_days >= required_days:
        print("Запрос на отпуск одобрен.")
    else:
        print("Недостаточно доступных дней для отпуска.")
else:
    print("Отпуск не запрашивался.")
```

3. Проверка данных на форме регистрации

Система проверяет, заполнены ли основные поля формы регистрации, а затем — правильно ли указан email.

Python
```
name_filled = True
email_filled = True
email_valid = False

if name_filled:
    print("Имя заполнено.")
    if email_filled:
        print("Email заполнен.")
```

```
        if email_valid:
            print("Email действителен.")
        else:
            print("Email недействителен.")
    else:
        print("Email не заполнен.")
else:
    print("Имя не заполнено.")
```

IT CAREER HUB

Равносильный код с вложенностью и без

Можно написать код со вложенными условиями, а также равносильный ему код без вложенности, используя логические операторы `and` и `or`. Код без вложенности предоставляет более линейный и компактный подход, уменьшая количество уровней вложенности.

Вариант с вложенностью

```python
age_valid = False

region_allowed = False

special_permission = False



if age_valid:

    if region_allowed or special_permission:

        print("Доступ разрешен.")

    else:

        print("Ваш регион не поддерживается.")

else:

    print("Возраст не соответствует требованиям.")
```

Вариант без вложенности

```python
age_valid = False

region_allowed = False
```

```
special_permission = False


if not age_valid:
    print("Возраст не соответствует требованиям.")
elif not (region_allowed or special_permission):
    print("Ваш регион не поддерживается.")
else:
    print("Доступ разрешен.")
```

IT CAREER HUB

Задание для закрепления 2

Какой результат выведет следующий код?

```python
x = 7

if x > 8:

    if x < 10:

        print("x меньше 10")

    else:

        print("x больше или равно 10")
```

a. x больше 5 и меньше 10
b. x больше 5, но не меньше 10
c. Ошибка
d. Программа ничего не выведет

Посмотреть ответ

Тернарный оператор

Тернарный оператор — это специальная конструкция, которая позволяет выполнять условную операцию на одной строке кода. Он является краткой формой записи оператора if-else, когда нужно присвоить значение или выполнить действие в зависимости от условия. Это делает код более компактным и читабельным.

Синтаксис тернарного оператора:

```python
<выражение_если_True> if <условие> else <выражение_если_False>
```

- `<условие>` — это выражение, которое проверяется на истинность.
- `<выражение_если_True>` — выполняется или возвращается, если условие истинно.
- `<выражение_если_False>` — выполняется или возвращается, если условие ложно.

Пример

```python
age = 18

status = "Взрослый" if age >= 18 else "Несовершеннолетний"

print(status)

x = 10

print("Чётное" if x % 2 == 0 else "Нечётное")
```

Преимущества тернарного оператора:

1. **Компактность**: Тернарный оператор позволяет сократить запись условий и сделать код более лаконичным, особенно когда нужно присвоить значение на основе простого условия.
2. **Читаемость**: Для простых проверок тернарный оператор делает код более читаемым, так как вся логика сосредоточена на одной строке.
3. **Использование в выражениях**: Тернарный оператор можно использовать внутри других выражений, что позволяет писать сложные конструкции в более сжатом виде.

Когда лучше НЕ использовать тернарный оператор?

Хотя тернарный оператор делает код компактным, его использование не всегда оправдано. Если условие или действия сложные и многозначные, лучше использовать стандартную конструкцию `if-else`, чтобы сохранить читаемость кода.

Пример

```python
x = 20

y = 10

z = 0

# Сложное условие лучше не помещать в тернарный оператор

print(x) if (x > 10 and y < 5 or z == 0) else print(0)
```

Тернарный оператор с несколькими условиями

Тернарный оператор в Python обычно используется для выполнения простых условий. Однако можно использовать тернарный оператор для обработки **нескольких условий**, вложив его в другой тернарный оператор. Такой подход позволяет решить более сложные задачи на одной строке, хотя следует помнить, что это может снизить читаемость кода.

Синтаксис с несколькими условиями:

```python
<выражение_если_условие1_True> if <условие1> else
<выражение_если_условие2_True> if <условие2> else <выражение_если_False>
```

Пример

Допустим, нужно определить оценку студента в зависимости от его баллов: если больше 90 — "Отлично", больше 75 — "Хорошо", больше 50 — "Удовлетворительно", иначе — "Неудовлетворительно".

```python
score = 80


grade = "Отлично" if score > 90 else "Хорошо" if score > 75 else "Удовлетворительно" if score > 50 else "Неудовлетворительно"

print(grade)
```

IT CAREER HUB

Конструкция match-case

Конструкция `match-case` была введена в Python 3.10 и позволяет сравнивать значения с шаблонами и выполнять соответствующие блоки кода. Конструкция значительно упрощает работу с множеством условий, особенно когда требуется проверить одно значение на соответствие нескольким возможным вариантам.

Синтаксис конструкции `match-case`:

```python
match <выражение>:

    case <шаблон1>:

        # код для выполнения, если выражение соответствует шаблону1

    case <шаблон2>:

        # код для выполнения, если выражение соответствует шаблону2

    case _:

        # код для выполнения, если не совпал ни один шаблон (аналог default)
```

- <выражение> — это переменная или выражение, которое будет проверяться.
- <шаблон> — значение или тип данных, с которым будет сравниваться выражение.
- `_` — это подстановочный символ, который используется, если ни одно из условий не совпало (аналог `else`).

Пример

```python
number = 7
```

```
match number:
    case 1:
        print("Это один.")
    case 2 | 3: # Символ "|" является аналогом оператора "or"
        print("Это два или три.")
    case _ if number > 5:
        print("Число больше 5.")
    case _:
        print("Это число меньше или равно 5.")
```

Здесь проверяются не только конкретные числа, но и условия с помощью `case _ if`, что позволяет более гибко управлять логикой программы.

IT CAREER HUB

Работа с типами данных

Конструкция `match-case` может проверять не только значения, но и типы данных.

```python
value = "Hello"



match value:

    case int():

        print("Это целое число.")

    case str():

        print("Это строка.")

    case bool():

        print("Это логический тип.")

    case _:

        print("Неизвестный тип данных.")
```

Здесь проверяется тип данных переменной `value`, и соответствующий блок кода выполняется в зависимости от типа.

Особенности конструкции `match-case`:

- **Четкость и лаконичность**: Конструкция `match-case` позволяет избегать множества операторов `if-elif`, делая код более лаконичным.
- **Поддержка шаблонов**: Можно использовать разные типы данных и шаблоны для проверки, включая не только значения, но и условия (`case _ if`).
- **Обработка нескольких вариантов через `|`**: Можно указать несколько вариантов в одном блоке case, разделяя их с помощью символа `|`.

IT CAREER HUB

Задание для закрепления 3

1. **Какой результат выведет следующий код?**

```python
temperature = 30

status = "Жарко" if temperature > 25 else "Прохладно"

print(status)
```

a. Жарко
b. Прохладно
c. Ошибка
d. Программа ничего не выведет

Посмотреть ответ

2. **Какой результат выведет следующий код?**

```python
number = 2

match number:

    case 1 | 2 | 3:

        print("Это один, два или три")

    case _ if number > 5:

        print("Число больше 5")

    case _:
```

```
        print("Число меньше или равно 5")
```

a. Это один, два или три
b. Число больше 5
c. Число меньше или равно 5
d. Ошибка

Посмотреть ответ

IT CAREER HUB

Ответы на задания

Задания на закрепление 1	Вернуться к заданиям
1. Результат выполнения кода	Ответ: b
2. Результат выполнения кода	Ответ: a, b
Задания на закрепление 2	Вернуться к заданиям
Результат выполнения кода	Ответ: d
Задания на закрепление 3	Вернуться к заданиям
1. Результат выполнения кода	Ответ: a
2. Результат выполнения кода	Ответ: a

IT CAREER HUB

Практическая работа

1. Знак числа

Напишите программу, которая получит число от пользователя и выведет, является ли это число положительным, отрицательным или равным нулю.

Пример вывода:

```
Python
Введите число: -3
Число отрицательное.

Введите число: 0
Число равно нулю.
```

Решение:

```
Python
num = int(input("Введите число: "))

if num > 0:
    print("Число положительное.")
elif num < 0:
    print("Число отрицательное.")
else:
    print("Число равно нулю.")
```

2. Стоимость билета

Напишите программу для контроля билетов на мероприятие. Пользователь вводит тип билета (стандартный, vip) и возраст. Условия:

- Дети до 12 лет: скидка 50% на все билеты.
- Взрослые (от 12 до 60 лет): полная стоимость.
- Пожилые (старше 60 лет): скидка 30% только на стандартные билеты.

Цены билетов:

- стандартный: 1000.
- vip: 3000.

Пример вывода:

```
Python
Введите тип билета (стандартный/vip): стандартный
Введите возраст: 65
Стоимость билета: 700.0
```

Решение:

```python
ticket_type = input("Введите тип билета (стандартный/vip): ")
age = int(input("Введите возраст: "))

if ticket_type == "vip":
    price = 3000
else:
    price = 1000

if age < 12:
    price *= 0.50
elif age > 60 and ticket_type == "стандартный":
    price *= 0.70

print("Стоимость билета:", price)
```