Лекция 11 – Использование условных операторов (if, elif, else) в Python для управления потоком выполнения программы

Условные операторы в Python — это фундаментальные инструменты программирования, которые позволяют выполнять различные блоки кода в зависимости от выполнения определенных условий. Они являются основой для создания логики в программах и принятия решений на основе данных.

Основные условные операторы Python

Оператор if – базовая проверка условий

Оператор if является основным условным оператором в Python. Он проверяет условие и выполняет блок кода только в том случае, если условие истинно (True).

Синтаксис:

```
if условие:
    # блок кода, который выполняется при истинном условии
```

Пример использования:

```
x = 10
if x > 5:
    print("x больше 5")  # Этот блок выполнится, потому что 10 > 5
```

Оператор elif — множественные условия

Оператор elif (сокращение от "else if") позволяет проверять несколько условий последовательно. Если первое условие if ложно, программа проверяет условие elif. Можно использовать несколько elif в одной конструкции.

Пример с elif:

```
x = 5
if x > 10:
    print("x больше 10")
elif x > 5:
    print("x больше 5")
elif x == 5:
    print("x равно 5")  # Этот блок выполнится
else:
    print("x меньше 5")
```

Оператор else – альтернативное выполнение

Оператор else выполняется только в том случае, если все предыдущие условия if и elif оказались ложными.

Пример с else:

```
x = 2
if x > 5:
    print("x больше 5")
else:
    print("x меньше или равно 5")  # Этот блок выполнится
```

Однострочные условия

Python позволяет записывать простые условия в одну строку, однако это может снижать читаемость кода:

```
x = 10
if x > 5: print("x больше 5")
```

Операторы сравнения в Python

Операторы сравнения возвращают логические значения True или False и используются для создания условий.

Основные операторы сравнения

Равенство (==) — проверяет равенство значений

```
x = 5
y = 5
print(x == y)  # True
```

Неравенство (!=) — проверяет неравенство значений

```
x = 5
y = 10
print(x != y)  # True
```

Больше (>) — проверяет, больше ли левый операнд правого

```
x = 10
y = 5
print(x > y)  # True
```

Меньше (<) — проверяет, меньше ли левый операнд правого

```
x = 5
y = 10
print(x < y)  # True
```

Больше или равно (>=) — проверяет, больше или равен ли левый операнд правому

```
x = 10
y = 10
print(x >= y)  # True
```

Меньше или равно (<=) — проверяет, меньше или равен ли левый операнд правому

```
x = 5
y = 10
print(x <= y)  # True
```

Логические операторы для сложных условий

Оператор and — логическое И

Возвращает True только когда оба условия истинны:

```
x = 5
y = 10
if x > 0 and y < 20:
    print("Оба условия истинны")  # Выполнится
```

Оператор or — логическое ИЛИ

Возвращает True если хотя бы одно условие истинно:

```
x = 5
y = 25
if x > 10 or y > 20:
    print("Хотя бы одно условие истинно")  # Выполнится
```

Оператор not — логическое НЕ

Инвертирует логическое значение:

```
x = 5
if not x == 10:
    print("x не равно 10")  # Выполнится
```

Комбинирование логических операторов

```
x = 15
if x > 0 and x < 20:
    print("x находится в диапазоне от 0 до 20")

if x < 0 or x > 20:
    print("x находится вне диапазона")
else:
    print("x в допустимом диапазоне")
```

Продвинутые техники работы с условиями

Множественные условия

Комбинирование нескольких условий для создания сложной логики:

```
age = 25
income = 50000
credit_score = 750

if age >= 18 and income > 30000 and credit_score > 650:
    print("Кредит одобрен")
elif age >= 18 and (income > 40000 or credit_score > 700):
    print("Кредит одобрен с условиями")
else:
    print("Кредит отклонен")
```

Вложенные условия

Создание условий внутри других условий для более детальной проверки:

```
x = 10
if x > 5:
    if x < 20:
        print("x больше 5 и меньше 20")
    else:
        print("x больше 5, но не меньше 20")
else:
    print("x меньше или равно 5")
```

Проверка принадлежности к коллекции

```
fruits = ["яблоко", "банан", "апельсин"]
fruit = "яблоко"

if fruit in fruits:
    print(f"{fruit} есть в списке")
else:
    print(f"{fruit} нет в списке")
```

Тернарный оператор — условия в одну строку

Тернарный оператор позволяет записать простое условие в одну строку, что делает код более компактным.

Синтаксис тернарного оператора

```
значение_если_истина if условие else значение_если_ложь
```

Примеры использования

Базовый пример:

```
x = 10
result = "положительное" if x > 0 else "отрицательное"
print(result)  # положительное
```

Присвоение значений:

```
age = 20
status = "взрослый" if age >= 18 else "несовершеннолетний"
print(status)  # взрослый
```

Вложенные тернарные операторы:

```
x = 0
result = "положительное" if x > 0 else "отрицательное" if x < 0 else
"ноль"
print(result)  # ноль
```

Практические примеры и лучшие практики

Валидация данных

```
def validate_email(email):
    if "@" in email and "." in email:
        return "Email корректен"
    else:
        return "Email некорректен"

# Использование
email = "user@example.com"
print(validate_email(email))
```

Работа с числовыми диапазонами

```
def check_grade(score):
    if score >= 90:
        return "Отлично"
    elif score >= 80:
        return "Хорошо"
    elif score >= 70:
        return "Удовлетворительно"
    elif score >= 60:
        return "Зачет"
```

```
    else:
        return "Незачет"

print(check_grade(85))  # Хорошо
```

Обработка исключительных случаев

```
def safe_divide(a, b):
    if b == 0:
        return "Деление на ноль невозможно"
    else:
        return a / b

result = safe_divide(10, 2)  # 5.0
result = safe_divide(10, 0)  # Деление на ноль невозможно
```

Советы по оптимизации условий

Используйте наиболее вероятные условия первыми для повышения производительности

```
Избегайте глубокой вложенности — используйте elif вместо множественных
вложенных if
```

Применяйте тернарный оператор для простых условий для улучшения читаемости

Группируйте связанные условия с помощью скобок для ясности

Используйте описательные имена переменных для понимания логики условий

Условные операторы в Python являются мощным инструментом для создания динамических и интеллектуальных программ. Понимание их работы и правильное применение поможет вам писать более эффективный и читаемый код.