# Условные операторы

Условные операторы – это конструкции в языках программирования, которые позволяют выполнить определённый код в зависимости от того, истинно или ложно некоторое условие. В Python существуют следующие условные операторы:

1. **if-else**: Это базовая конструкция, которая позволяет выполнить код, если условие истинно (if), и другой код, если условие ложно (else).

if условие:

# Код, если условие истинно

else:

# Код, если условие ложно

1. **elif**: Это расширение if, которое позволяет проверить несколько условий по очереди.

if первое\_условие:

# Код, если первое условие истинно

elif второе\_условие:

# Код, если второе условие истинно

else:

# Код, если ни одно из условий не истинно

1. **match-case**: Это новая конструкция, появившаяся в Python 3.10, которая похожа на switch-case в других языках программирования. Она позволяет сопоставить значение с несколькими шаблонами и выполнить код для соответствующего шаблона.

match значение:

case шаблон1:

# Код для шаблона1

case шаблон2:

# Код для шаблона2

case \_:

# Код, если ни один шаблон не совпал

# Разница между if-else и match-case.

Представим, что у нас есть коробка с фруктами. Если ты откроешь коробку и увидишь яблоко, ты скажешь: "Это яблоко". А если там не яблоко, ты скажешь: "Это не яблоко". Это как **if-else** в Python. Ты проверяешь, яблоко это или нет, и говоришь что-то в зависимости от ответа.

Теперь представим, что у нас есть большая коробка с разными фруктами. Если ты откроешь коробку и увидишь яблоко, ты скажешь: "Это яблоко". Если ты увидишь банан, ты скажешь: "Это банан". И если ты увидишь апельсин, ты скажешь: "Это апельсин". И так далее для каждого фрукта. Это как **match-case** в Python. Ты проверяешь каждый фрукт и говоришь что-то для каждого из них.

Так что **if-else** используется, когда у тебя два варианта: что-то либо верно, либо нет. А **match-case** используется, когда у тебя много разных вариантов, и ты хочешь проверить их все.

Можно использовать **if-elif-else** для тех же целей, что и **match-case**. Разница в том, что **match-case** делает код более читаемым и понятным, когда у тебя много условий. Это как если бы у тебя было много разных ключей, и ты хочешь легко найти правильный ключ для каждой двери, не пробуя каждый ключ по очереди.

Вот пример с ключами:

# if-elif-else

ключ = "золотой"

if ключ == "золотой":

print("Открывает золотую дверь")

elif ключ == "серебряный":

print("Открывает серебряную дверь")

elif ключ == "бронзовый":

print("Открывает бронзовую дверь")

else:

print("Ключ не подходит")

# match-case

ключ = "золотой"

match ключ:

case "золотой":

print("Открывает золотую дверь")

case "серебряный":

print("Открывает серебряную дверь")

case "бронзовый":

print("Открывает бронзовую дверь")

case \_:

print("Ключ не подходит")

Как видишь, **match-case** выглядит аккуратнее и проще для понимания, особенно когда у тебя много разных вариантов. Это помогает программистам быстрее понять, что делает код, и легче его изменять. Но если у тебя всего несколько условий, **if-elif-else** тоже хорошо работает.

# Способы реализации ветвления.

Python есть несколько способов реализации ветвления, помимо **if-else** и **match-case**. Вот некоторые из них:

1. **Использование словаря для замены switch-case:**  
   Вместо множества if-elif-else, можно использовать словарь, где ключи - это условия, а значения - это функции, которые нужно выполнить.

def золотая\_дверь():

print("Открывает золотую дверь")

def серебряная\_дверь():

print("Открывает серебряную дверь")

def бронзовая\_дверь():

print("Открывает бронзовую дверь")

ключ\_действия = {

"золотой": золотая\_дверь,

"серебряный": серебряная\_дверь,

"бронзовый": бронзовая\_дверь

}

ключ = "золотой"

# Вызов функции, соответствующей ключу

ключ\_действия.get(ключ, lambda: print("Ключ не подходит"))()

1. **Использование лямбда-функций в словаре:**  
   Можно также непосредственно определить лямбда-функции в словаре, если действия короткие и не требуют отдельного определения функции.

ключ\_действия = {

"золотой": lambda: print("Открывает золотую дверь"),

"серебряный": lambda: print("Открывает серебряную дверь"),

"бронзовый": lambda: print("Открывает бронзовую дверь")

}

ключ = "золотой"

ключ\_действия.get(ключ, lambda: print("Ключ не подходит"))()