

Требуется построить таблицу истинности для формулы:

$$F = ((A \wedge (B \rightarrow \bar{C})) \wedge (\bar{B} \rightarrow A) \vee B).$$

Решение. Определим порядок действий и заполним последовательно таблицу истинности. Имеем:

$$F = \left(\left(A \wedge \underbrace{\left(B \rightarrow \underbrace{\bar{C}}_1 \right)}_2 \right)_3 \wedge \underbrace{\left(\bar{B} \rightarrow A \right)}_5 \right)_6 \vee B, \quad \underbrace{\hspace{10em}}_7$$

Т.е. у нас 2 скобки со скобками, внутренние в приоритете, однако почему в левом выражении скобки мы начинаем считать с правой части №1, а в правом - №4 у нас начинается с левой, т.е. дальше логика последовательности вычисления мне понятна вполне. Но с шагом 1 и шагом 4 - не очень.

Для ответа на этот вопрос вспоминаем приоритеты логических операций: not, and, or, xor, if...then, equal. Операция not может записываться черточкой над операндом. С учетом скобочек и приоритетов операций и выполняется построение таблицы истинности.

<https://apipython.ru/prioritetnost-i-assocziativnost-operatorov-v-python-ischerpyvayushhee-obyasnenie/>

1. Нужно повторить и запомнить приоритеты логических операций: операции or и and выполняются в последнюю очередь. Операции in, is, not и операции сравнения выполняются в приоритетном порядке

Итак, подведем итоги по блоку логики. Логические задачи - это задачи на определение попадания определенной сущности в какой-либо интервал (временной, числовой), попадания в

какую-либо категорию (большой-средний-малый, красный-белый-зеленый и т.д. и т.п.) или соответствия какому-либо признаку. Логические задачи можно разделить на 2 категории: ---

Задачи с простой (линейной) логикой ---

Задачи с вложенной логикой.

Например, к задачам с **линейной логикой** можно отнести задачу с ранжированием резюме по дате их составления.

К задачам с вложенной логикой можно отнести задачу проверки делимости числа на 2 и на 5, когда внутри проверки делимости на 2 нужно еще проверить и делимость на 5.

Задачи с линейной логикой. Задачи с линейной логикой можно разделить на задачи унарной логики, задачи бинарной логики и задачи множественного выбора.

Задачи унарной логики просты: они разбивают перспективу выбора на 2 части - либо это, либо то.

Задачи бинарного выбора делит перспективу выбора на 3 части. К нему относится уже пройденная задача ранжирования резюме. Методика решения таких задач проста: для временных и числовых интервалов проверяем попадание сущности в крайние интервалы, в противном случае она попадает в средний интервал. Проверку на категории проводим последовательно, слева направо.

Задачи множественного выбора разбивают перспективу на множество (не менее 4-х) интервалов выбора. Такие задачи тупо решаем перебором вариантов выбора слева направо. Для решения таких задач лучше всего использовать оператор множественного выбора Match Case. К сожалению, разработчики Python не читают в детстве русские народные сказки и с перспективой множественного выбора знакомятся уже в зрелом возрасте. По этой причине, видимо, оператор Match Case впервые реализован в 10-й версии Python, что нам в учебных материалах не доступно. Ознакомится можно здесь <https://pythonist.ru/konstrukciya-match-case-v-python-polnoe-rukovodstvo/?ysclid=lpbi51gyqx748347853>.

Задачи с вложенной логикой. Данные задачи решаются методом декомпозиции. Цель декомпозиции - получить наружный слой решения задачи в виде линейной логики. Далее, начинаем разбираться внутри каждого линейного интервала и снова классифицировать часть задачи. Если задача получилась с линейной логикой, то решаем ее по известным методикам, иначе повторяем методику работы с задачами с вложенной логикой.

В заключение хочу сказать, что существует класс задач с "бухгалтерской" логикой. Это задачи с противоречивой логикой и могут не иметь формального решения. Самая большая сложность - это сложность классификации логической задачи. Здесь помогут только опыт и сообразительность, коллеги и наставники.



pythonist.ru

[Конструкция match-case в Python](#)

Полное руководство по использованию выражений match-case в Python. Разобраны основные шаблоны сравнений, приведен код примеров.

КМ: вот тут не понятно, как 4ка совпадает с подстановкой, если ее в принципе быть не может. Т.е. - подстановка тут - любое возможное значение, кроме орла и решки, так? Т.е. получается - мы проверяем правильные (реально возможные ответы) - а все остальное закидываем в подстановку - "...задачи тупо решаем перебором вариантов выбора слева направо..."

```
coinflip = 4
match coinflip:
    case 1:
        print("Heads")
        exit()
    case 0:
        print("Tails")
    case _:
        print("Must be 0 or 1.")
```

Результат:

```
Must be 0 or 1.
```

Здесь знак подстановки соответствует всему кроме 0 или 1. В нашем коде результат подбрасывания монеты в переменной `coinflip` по какой-то причине равен 4. Этот результат совпадает с подстановкой и поэтому запускается соответствующее действие.

ВВ: А вот здесь у нас все очень просто: для единицы - орел, для нуля - решка, а для всего остального - будет не определено, либо то либо это. Оператор case обрабатывает 4 в секции с подчеркиванием как "все остальные результаты".

1

```
location = (0, 0)
match location:
    case _:
        print("1D location found")
    case (_, _):
        print("2D location found")
    case (_, _, _):
        print("3D location found")
```

Результат:

2D location found

Как можно заметить, в этом случае проверяется только количество найденных элементов. Здесь мы использовали подстановку, чтобы убедиться, что в кортеже есть два элемента, не заботясь об отдельных значениях элементов.

То же самое можно сделать, используя оператор if-else и функцию `len()`. Но если операторы if-else становятся слишком длинными, лучше использовать оператор match-case для повышения качества кода.

ВВ: А в этом примере case работает по количеству элементов в кортеже. При этом подчеркивание в case говорит о том, что само значение элемента кортежа может быть любым, т.е. тем самым "все остальные".

КМ: 1) не понятно. Из моего сложившегося представления - это инфа из какого-то "хранилища", но при этом - когда попадают эти выражения - я подсудно понимаю, что это "хранилище" - вот тут же ниже и создаем. диссонанс понимания. 2) вот эта связка не понятна. как оно соединяет названия и цифры - куда и для чего вообще тут цифры значений, если текстово мы пользуемся названиями только

В качестве шаблонов выражения match-case можно использовать элементы перечислений (enumerations).

Чтобы это продемонстрировать, давайте создадим класс перечисления `Direction` (унаследованный от класса `Enum`), представляющий четыре основных направления на компасе.

Далее создадим функцию `handle_directions()`, которая принимает элемент класса `Direction` в качестве входного аргумента. Эта функция сопоставляет этот аргумент с одним из направлений из перечисления и реагирует соответствующим образом.

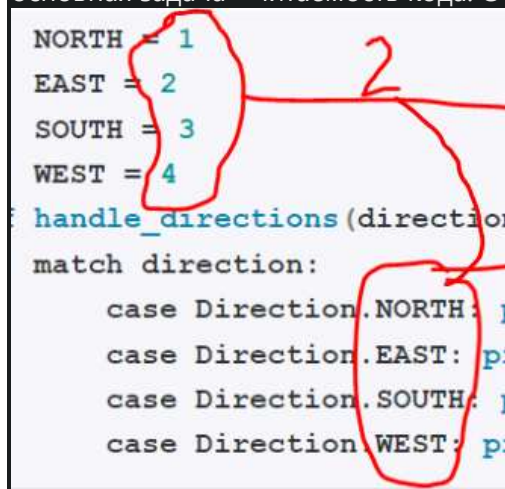
Вот сам код:

```
1 from enum import Enum

class Direction(Enum):
    NORTH = 1
    EAST = 2
    SOUTH = 3
    WEST = 4
2
3 def handle_directions(direction):
4     match direction:
5         case Direction.NORTH: print("Heading North")
6         case Direction.EAST: print("Heading East")
7         case Direction.SOUTH: print("Heading South")
8         case Direction.WEST: print("Heading West")
```

ВВ: Здесь пример более сложный, ибо мы вторгаемся в объектно-ориентированное программирование. Скажем так, у нас описан некий класс направлений. Класс - это некое описание определенного множества сущностей. Например, класс "человек" - туловище, две ноги, две руки, голова, хвост. А нет, хвоста нет. А вот экземпляры этого самого класса человек это я, Вы, еще кто-нибудь, кто похож на человека. Класс - это описание, и в нашем случае содержит описание 4-х направлений и нумерует их, так он наследник от перечислимого типа (т.е. типа, который можно посчитать, например как в игре в прятки, раз, два, три, четыре, пять...). А вот дальше в блоке match-case мы начинаем реализовывать варианты, в зависимости от выбора направления, обозначенного в переменной direction. В этом случае переменная direction содержит в себе какое-нибудь число, соответствующее одному из направлений. Собственно, в блоке выбора можно варианты выбора написать и так: case 1:..., case 2:... и т.д. Это тоже верно. Но нам нужен читаемый код, мы должны понимать, что мы выбираем, поэтому цифири заменили на названия направлений, описанные в классе Direction. Такая техника тоже допустима. Просто при сравнении вариантов в этом случае так же подставится число, соответствующее буквенному обозначению направления. Получаются следующие эквивалентные выражения: direction = 1 и direction = Direction.Nothing и т.д. И да, к атрибутам (свойствам) класса обращение идет через точку, как и к методам.

КМ: давайте теперь тут. у меня вчера интернет на пол дня заглушили, но статью я дочитала. Дальше вроде явных непоняток не появилось. Здесь:...**в блоке выбора можно варианты выбора написать и так: case 1:..., case 2:...поэтому цифири заменили на названия...при сравнении вариантов в этом случае так же подставится число, соответствующее буквенному обозначению направления...direction = 1 и direction = Direction.Nothing... => ну вот, перечитала с 4 раза, пока вопрос формулировала - похоже сама поняла, про эквивалентность -ок. А сейчас тупой вопрос - для чего мы обозначаем тогда цифрами, если основная задача - читаемость кода. С классами тоже все вразумительно, спасибо!**



```
NORTH = 1
EAST = 2
SOUTH = 3
WEST = 4

handle_directions(direction)
match direction:
    case Direction.NORTH:
    case Direction.EAST:
    case Direction.SOUTH:
    case Direction.WEST:
```

ВВ: Хорошо, пойдём дальше в дебри ООП, потом легче будет. Итак, любой класс может иметь или не иметь атрибуты, т.е. какие-то объявленные переменные. Есть еще один небольшой секрет: атрибуты класса могут генерироваться без объявления при создании экземпляров

класса. Пока этот механизм оставим в покое. В классе `Direction` описаны (объявлены) 4 его атрибута, а именно именно `NORTH`, `EAST`, `SOUTH`, `WEST`. Эти атрибуты есть не что иное как имена переменных. Но далее в описании класса этим переменным присваивается значение и они становятся константами и всегда имеют одно и то же значение. Попытка изменения их значений в любом другом месте кода приведет к исключению. Именно из-за неизменности значений эти атрибутов мы и можем использовать их буквенную нотацию в операторе `match-case`. При исполнении программы интерпретатор, дойдя до первой секции `case` и встретив конструкцию `Direction.NOTH`, полезет в табличку с описанием классов, найдет класс `Direction`, в нем найдет атрибут `NOTH` и подставит вместо него `1` и будет сравнивать значение переменной `direction` с `1` и далее аналогично по списку. При совпадении с каким-либо вариантом будет выведено соответствующее сообщение.

ВВ:

```
from enum import Enum
```

Это запись означает, что описание класса `Enum` мы возьмем из модуля с именем `enum`. Т.е. исходный код описания класса `Enum` грубо говоря мы найдем в файлике с именем `enum.py`.
Доброе утро! Начну с простого. Вот тут "мы найдем в файлике с именем `enum.py`" - именно такое понимание фразы у меня есть, НО - а файл где? т.е. он каким то образом крепится/данные как transferятся в код? видимо заморочилась, первички нет, так сказать. Потому у меня постоянный затык с этим высказыванием.

ВВ:

Заморачиваться с тем, в каком каталоге находится файл, право, не стоит. Когда дойдем до установки Python, там разъясню подробнее. Но для общего образования сообщу, что прозрачные каталоги в винде описываются переменной `path`. В ней и записаны прозрачные каталоги Python. И в самой настройке интерпретатора в специальных системных файлах указаны все файлы, доступные для оперированию интерпретатором.

1

Тут нужно запомнить прямое правило: все, что нужно, Python найдет сам. А если не найдет, то сообщит об этом. И вот тогда и будем скакать

КМ: Ага, вот теперь вроде прояснилось. Просто надо научиться в т.ч. не только буквально читать слово, а воспринимать его как переменную со значением=константу

ВВ: Было бы значительно проще все воспринимать, если бы было понятие как работает интерпретатор.

ВВ: Еще немного ассоциаций. Вспомним русскую сказку с множественным выбором: налево пойдешь - коня потеряешь, направо пойдешь - голову потеряешь, прямо пойдешь - змей-горыныча встретишь. Варианты выбора в этом случае описаны словами. Но ведь мы можем на камне нарисовать стрелки и поместить соответствующие значки. Например, перечеркнутого коня, отрубленную голову, змей-горыныча. Так и здесь: цифры - это слова, а атрибуты класса - это значки. **Суть выбора от этого не меняется.**