Вот пример работы с циклом, написанный на JavaScript:

```
let scores = [54,67,48,99,27];
for(const i=0; i < scores.length; i++) {
  console.log(i, scores[i]);
}
```

В рассмотренном примере переменная i не имеет какого-то явного отношения к массиву scores. Она всего лишь хранит некое число, которое увеличивается на 1 в каждой итерации цикла, и которое, как оказывается, подходит для последовательного обращения к элементам массива по их индексам.

Но надо отметить, что в JavaScript существуют методы, позволяющие перебирать массивы, так сказать, без посредников. Речь идёт о циклах forEach и for of.

Цикл for в Python использует итерацию на основе коллекции. Это означает, что Python на каждой итерации назначает следующий элемент из iterable переменной цикла, как в этом примере:

```
values = ["a", "b", "c"]
for value in values:
     print(value)
```

Преимущество такой итерации заключается в том, что она помогает избежать ошибки off-by-one, которая часто встречается в других ЯП.

Теперь представим, что нам необходимо вывести в списке индекс элемента на каждой итерации. Один из способов решения – создать переменную для хранения индекса и обновлять ее на каждой итерации:

```
index = 0
for value in values:
     print(index, value)
     index += 1
```

Другой распространенный способ решения этой проблемы – использовать range() в сочетании с len() для автоматического создания index, и вам не придется помнить о его обновлении:

```
scores = [54,67,48,99,27]
for i in range(len(scores)):
  print(i, scores[i])
```

Проблема этого цикла заключается в том, что он не очень хорошо соответствует идеологии Python. В нём мы не перебираем список, а, вместо этого, используем вспомогательную переменную i для обращения к элементам списка.В цикле for можно использовать любую итерацию, но только последовательности доступны по целочисленным индексам.

# Использование enumerate()

Если вам нужно адекватным образом отслеживать «индекс элемента» в for-цикле Python, то для этого может подойти функция enumerate(), которая позволяет «пересчитать» итерируемый объект. Её можно использовать не только для обработки списков, но и для работы с другими типами данных — со строками, кортежами, словарями. Она доступна в Python с версии 2.3.

Синтаксис функции enumerate() выглядит следующим образом:

enumerate(iterable, start)

Функция enumerate() принимает два параметра: iterable и start.

- iterable — это итерируемый объект (список, кортеж и т.д.), который будет возвращен в виде пронумерованного объекта (объекта enumerate)
- start — это начальный индекс для возвращаемого объекта enumerate. Значение по умолчанию равно 0, поэтому, если вы опустите этот параметр, в качестве первого индекса будет использоваться 0.

## Пример №1: функция enumerate() с одним параметром

```
names = ["John", "Jane", "Doe"]

enumNames = enumerate(names)

print(list(enumNames))

# [(0, 'John'), (1, 'Jane'), (2, 'Doe')]
```

В приведенном выше примере мы создали список, состоящий из трех имен.

Затем мы преобразовали переменную names в объект enumerate и сохранили результат в переменной с именем enumNames.

Далее мы захотели, чтобы наш объект был выведен в виде списка, поэтому мы сделали следующее: list(enumNames).

При выводе на консоль результат будет выглядеть так: [(0, 'John'), (1, 'Jane'), (2, 'Doe')].

Как видите, результат мы получили в виде пар ключ-значение. Первый индекс — 0, он связан с первым элементом в нашем списке имен names, второй — 1, связан со вторым элементом в списке names и так далее.

Обратим внимание, что в нашем примере мы использовали только первый параметр функции enumerate().

В следующем примере мы используем оба параметра, чтобы вы увидели разницу.

## Пример №2: функция enumerate() с указанием начального индекса

```
names = ["John", "Jane", "Doe"]

enumNames = enumerate(names, 10)

print(list(enumNames))

# [(10, 'John'), (11, 'Jane'), (12, 'Doe')]
```

Здесь мы добавили второй параметр в функцию enumerate(), написав enumerate(names, 10).

Значение второго параметра, 10, будет начальным индексом для ключей (индексов) в объекте enumerate.

Результат будет таким: [(10, 'John'), (11, 'Jane'), (12, 'Doe')].

## Как перебрать результат функции enumerate() в Python

```
for count, value in enumerate(values):
   print(count, value)


for count, value in enumerate(values, start=10):
   print(count, value)
```

# Понимание enumerate()

До этого мы рассматривали примеры использования enumerate(). Теперь стоит глубже изучить, как эта функция работает.

Чтобы лучше понять, как работает enumerate(), реализуйте собственную версию с помощью Python. Она должна следовать двум требованиям:

- принимать iterable и начальное значение в качестве аргументов;
- отправлять обратно кортеж с текущим значением счетчика и связанным с ним элементом из iterable.

Один из способов написания функции по данным спецификациям приведен в документации Python:

```
def my_enumerate(sequence, start=0):
```

```
    n = start
    for elem in sequence:
         yield n, elem
         n += 1
```

my_enumerate() принимает два аргумента: sequence и start. Значение по умолчанию start равно 0. Для каждого элемента в последовательности текущие значения n и elem отправляются обратно.

Таким образом, мы реализовали эквивалент enumerate() всего из нескольких строк кода, хотя оригинальный код на C для enumerate() несколько больше.

## Распаковка аргументов с помощью enumerate()

Когда вы используете enumerate() в цикле for, вы говорите Python работать с двумя переменными: одной для подсчета и одной для значения. Все это можно сделать, используя распаковку аргументов.

Когда вызывается enumerate() и передается последовательность значений, Python возвращает итератор, а когда вы запрашиваете у итератора следующее значение, он отдает кортеж с двумя элементами: элемент кортежа (счетчик) и значение из переданной последовательности.

```
values = ["a", "b"]
enum_instance = enumerate(values)
next(enum_instance)
(0, 'a')
next(enum_instance)
(1, 'b')
next(enum_instance) Traceback (most recent call last):   File
"<stdin>", line 1, in <module>
StopIteration
```

## Почему не имеет смысла перечислять словари и наборы

Итак, имеет ли смысл использовать функцию `enumerate` для словарей и наборов?

**Категорически нет!**

Подумайте об этом, единственная причина, по которой вы бы использовали `enumerate`, - это когда вы действительно заботитесь об индексе элемента.

Словари и наборы не являются последовательностями. Их элементы не имеют индекса, и он им, по определению, не нужен.

# Лучшие практики использования Enumerate()

### Ведение индексов

С enumerate(), нет необходимости ручного контроля индексов, что снижает вероятность ошибок.

### За читаемость кода

Использование enumerate() сделает код более чистым и понятным, так как он сокращает и структурирует общепринятые шаблоны, когда требуются и элементы, и их индексы.

### Совместимость на высшем уровне

Функция Enumerate() отлично сочетается с различными конструкциями Python, будь то if/else, генераторы или лямбда-функции.

### Высокая скорость исполнения

За счет реализации на языке C, enumerate() обеспечивает высокую производительность, что позволяет ему занимать достойное место в арсенале инструментов Python.