List comprehension — это упрощенный подход к созданию списка, который задействует цикл for, а также инструкции if-else для определения того, что в итоге окажется в финальном списке.

**Преимущества list comprehension**

У list comprehension есть три основных преимущества.

1. Простота. List comprehension позволяют избавиться от циклов for, а также делают код более понятным. В JavaScript, например, есть нечто похожее в виде map() и filter(), но новичками они воспринимаются сложнее.
2. Скорость. List comprehension быстрее for-циклов, которые он и заменяет. Это один из первых пунктов при рефакторинге Python-кода.
3. Принципы функционального программирования. Это не так важно для начинающих, но функциональное программирование — это подход, при котором изменяемые данные не меняются. Поскольку list comprehensions создают новый список, не меняя существующий, их можно отнести к функциональному программированию.

**Создание первого list comprehension**

List comprehension записывается в квадратных скобках и задействует цикл for. В процессе создается новый список, куда добавляются все элементы оригинального. По мере добавления элементов их можно изменять.

Для начала возьмем простейший пример: создадим список из цифр от 1 до 5, используя функцию range().

КОПИРОВАТЬ

>>> nums = [n for n in range(1,6)]

>>> print(nums)

[1, 2, 3, 4, 5]

В этом примере каждое значение диапазона присваивается переменной n. Каждое значение возвращается неизменным и добавляется в новый список. Это — та самая n перед циклом for.

В качестве итерируемого объекта не обязательно должна быть функция range(). Это может быть любое итерируемое значение.

**List comprehension с изменением**

Теперь пойдем чуть дальше и добавим изменение для каждого значения в цикле.

КОПИРОВАТЬ

>>> nums = [1, 2, 3, 4, 5]

>>> squares = [n\*n for n in nums]

>>> print(squares)

[1, 4, 9, 16, 25]

В этом примере два изменения по сравнению с прошлым кодом. Во-первых, в качестве источника используется уже существующий список. Во-вторых, list comprehension создает список, где каждое значение — это возведенное в квадрат значения оригинального списка.

**List comprehension с if**

Теперь добавим проверку с помощью if, чтобы не добавлять все значения.

КОПИРОВАТЬ

>>> nums = [1, 2, 3, 4, 5]

>>> odd\_squares = [n\*n for n in nums if n%2 == 1]

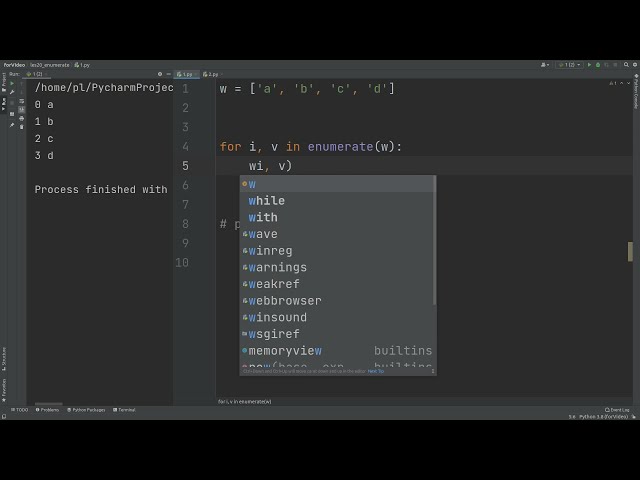
>>> print(odd\_squares)

[1, 9, 25]

# Функция enumerate

В Python есть еще одна встроенная функция, которая часто используется в заголовке **for**. Это функция enumerate(). Если range() позволяет получить только индексы элементов списка, то enumerate() – сразу индекс элемента и его значение.

Функция enumerate() применяется для так называемых итерируемых объектов (список относится к таковым) и создает объект-генератор, который генерирует кортежи, состоящие из двух элементов – индекса элемента и самого элемента.



>>> spisok = [16, 46, 26, 36]

>>> for i in enumerate(spisok):

... print(i)

...

(0, 16)

(1, 46)

(2, 26)

(3, 36)

>>> b = "hello"

>>> for i in enumerate(b):

... print(i)

...

(0, 'h')

(1, 'e')

(2, 'l')

(3, 'l')

(4, 'o')

Эти кортежи можно распаковывать, то есть извлекать индекс и значение, в теле цикла:

>>> for item in enumerate(spisok):

... print(item[0], item[1])

...

0 16

1 46

2 26

3 36

Однако чаще это делают еще в заголовке **for**, используя две переменные перед **in**:

>>> for id, val in enumerate(spisok):

... print(id, val)

...

0 16

1 46

2 26

3 36

**Функция enumerate() используется для упрощения прохода по коллекциям в цикле, когда кроме самих элементов требуется их индекс:**

>>> a = [10, 20, 30, 40]

>>> for id, item in enumerate(a):

... a[id] = item + 5

...

>>> a

[15, 25, 35, 45]

# Функция zip

В Pyhon функция zip позволяет пройтись одновременно по нескольким итерируемым объектам (спискам и др.):

>>> a = [10, 20, 30, 40]

>>> b = ['a', 'b', 'c', 'd', 'e']

>>> for i, j in zip(a, b):

... print(i, j)

...

10 a

20 b

30 c

40 d

Здесь выражение zip(a, b) создает объект-итератор, из которого при каждом обороте цикла извлекается кортеж, состоящий из двух элементов. Первый берется из списка a, второй - из b.

>>> for i in zip(a, b):

... print(i, type(i))

...

(10, 'a') <class 'tuple'>

(20, 'b') <class 'tuple'>

(30, 'c') <class 'tuple'>

(40, 'd') <class 'tuple'>

В списке b на один элемент больше. Функция zip возвращает итератор, который останавливается, когда исчерпывается самая короткая последовательность. Если требуется учесть все значения из самой длинной, то следует использовать функцию zip\_longest из модуля itertools:

>>> a = [10, 20, 30, 40]

>>> b = ['a', 'b', 'c', 'd', 'e']

>>> c = [1.1, 1.2]

>>> for i in itertools.zip\_longest(a,b,c):

... print(i)

...

(10, 'a', 1.1)

(20, 'b', 1.2)

(30, 'c', None)

(40, 'd', None)

(None, 'e', None)

Как мы видим, если элемента не хватает, то по-умолчанию подставляется объект None. Можно указать свой вариант заполнения:

>>> for i in itertools.zip\_longest(a,b,c, fillvalue=0):

... print(i)

...

(10, 'a', 1.1)

(20, 'b', 1.2)

(30, 'c', 0)

(40, 'd', 0)

(0, 'e', 0)

Если требуется получить не итератор, возвращаемый zip(), а список из элементов, то к объекту zip применима функция list, которая преобразует итератор в список:

>>> a = [10, 20, 30, 40]

>>> c = [1.1, 1.2, 1.3, 1.4]

>>> ac = zip(a, c)

>>> type(ac)

<class 'zip'>

>>> ac = list(ac)

>>> type(ac)

<class 'list'>

>>> ac

[(10, 1.1), (20, 1.2), (30, 1.3), (40, 1.4)]

При написании программ на Python функцию zip нередко применяют для выполнения совместных или одновременных действий над элементами разных списков.

>>> values = [1.34, 3.25, 2.99]

>>> coefficient = [3, 2, 2]

>>> for i, j in zip(values, coefficient):

... print(i\*j)

...

4.0200000000000005

6.5

5.98

>>> a = []

>>> b = []

>>> for i, j in zip(range(10,20), range(1,10)):

... a.append(i)

... b.append(j)

...

>>> a

[10, 11, 12, 13, 14, 15, 16, 17, 18]

>>> b

[1, 2, 3, 4, 5, 6, 7, 8, 9]