1. Variabel

Variabel adalah nama yang digunakan untuk menyimpan nilai. Anda dapat membuat variabel dengan format seperti berikut :

nama_variabel = nilai

Operator "=" dalam python digunakan untuk menetapkan nilai variabel yang berada di sebelah kiri. Nama variabel tidak menggunakan tanda petik.

```python
# Membuat variabel
bunga = 10
```

bunga merupakan variabel

10 merupakan nilai

2. Data Types & Conversion

Tipe Data

Tipe data adalah suatu media atau memori pada komputer yang digunakan untuk menampung informasi.

Python sendiri mempunyai tipe data yang cukup unik bila kita bandingkan dengan bahasa pemrograman yang lain.

Berikut adalah tipe data dari bahasa pemrograman Python

Tipe Data	Contoh	Penjelasan
Boolean	`True` atau `False`	Menyatakan benar `True` yang bernilai `1`, atau salah `False` yang bernilai `0`
String	`"Ayo belajar Python"`	Menyatakan karakter/kalimat bisa berupa huruf angka, dll (diapit tanda `"` atau `'`)
Integer	`25` atau `1209`	Menyatakan bilangan bulat
Float	`3.14` atau `0.99`	Menyatakan bilangan yang mempunyai koma
Hexadecimal	`9a` atau `1d3`	Menyatakan bilangan dalam format heksa (bilangan berbasis 16)
Complex	`1 + 5j`	Menyatakan pasangan angka real dan imajiner
List	`['xyz', 786, 2.23]`	Data untaian yang menyimpan berbagai tipe data dan isinya bisa diubah-ubah
Tuple	`('xyz', 768, 2.23)`	Data untaian yang menyimpan berbagai tipe data tapi isinya tidak bisa diubah
Dictionary	`{'nama': 'adi','id':2}`	Data untaian yang menyimpan berbagai tipe data berupa pasangan penunjuk dan nilai

Konversi

Python mendefinisikan fungsi konversi tipe untuk secara langsung mengonversi satu tipe data ke tipe lain yang berguna dalam pemrograman sehari-hari dan kompetitif.

Artikel ini ditujukan untuk memberikan informasi tentang fungsi konversi tertentu.

Ada dua jenis Jenis Konversi dengan Python:

1. Jenis Konversi Implisit
2. Konversi Jenis Eksplisit

Jenis Konversi Implisit

Dalam konversi tipe implisit dari tipe data dengan Python, interpreter Python secara otomatis mengonversi satu tipe data ke tipe data lainnya tanpa keterlibatan pengguna.

Untuk mendapatkan pandangan yang lebih jelas tentang topik, lihat contoh di bawah ini.

```python
x = 10
print("x is of type:", type(x))

y = 10.6
print("y is of type:", type(y))
```

```
x is of type: <class 'int'>
y is of type: <class 'float'>
```

```python
x = x + y

print(x)
print("x is type of:", type(x))
```

```
20.6
x is type of: <class 'float'>
```

Konversi Jenis Eksplisit

Dalam Konversi Tipe Eksplisit dengan Python, tipe data diubah secara manual oleh pengguna sesuai kebutuhan mereka. Berbagai bentuk konversi tipe eksplisit dijelaskan di bawah ini:

```python
# Python code to demonstrate Type conversion
# using int(), float()

# initializing string
s = "10010"

# printing string to int base 2
c = int(s,2)
print("After converting to interger base 2 : ", end="")
print(c)

# printing string converting to float
e = float(s)
print("After converting to float : ", end="")
print(e)
```

```
After converting to interger base 2 : 18
After converting to float : 10010.0
```

3. Creating List dan Tuple

List

List adalah struktur data pada python yang mampu menyimpan lebih dari satu data, seperti array. Anda dapat membuat list sebagai berikut ; [element 1, element2, ..]. Setiap nilai di dalam list disebut element. Dengan menggunakan list anda dapat mengelola nilai lebih dari satu tipe data.

Cara Membuat List

List dapat dibuat dengan menggunakan variabel, kemudian variabel tersebut didefinisikan dengan tanda kurung siku ([])

```python
# membuat list kosong
warna = []

# membuat list dengan 1 item
hobi = ["membaca"]
```

```python
# contoh lain list
buah = ["jeruk", "apel", "mangga", "duren"]
laci = ["buku", 21, True, 34.21]
```

```python
print(buah)
print(laci)
print("tipe data buah: ", type(buah))
print("tipe data laci: ", type(laci))
```

```
['jeruk', 'apel', 'mangga', 'duren']
['buku', 21, True, 34.21]
tipe data buah:  <class 'list'>
tipe data laci:  <class 'list'>
```

Tuple

```python
# Tuple biasanya dibuat dengan tanda kurung seperti ini:
t = (1234, 4321, 'Hello')
print(type(t))
```

```
<class 'tuple'>
```

```python
# atau bisa juga tanpa tanda kurung:
t = 1234, 4321, 'Hello'
print(type(t))
```

```
<class 'tuple'>
```

```python
# Apabila kita ingin membuat tuple tanpa isi, kita dapat menuliskan seperti ini:

# membuat tuple kosong
kosong = ()
print(type(kosong))
```

```
<class 'tuple'>
```

```python
# lalu untuk membuat Tuple yang hanya berisi satu (singleton), maka kita harus menambahkan tanda koma di akhir.
satu = ("isinya",)
print(type(satu))
```

```
<class 'tuple'>
```

Memotong Tuple

Sama seperti list, di Tuple juga kita bisa melakukan slicing. Contoh:

```python
# mula-mula kita punya tuple seperti ini:
web = (123, 'Petani Kode', 'https://www.petanikode.com')

# lalu kita ingin potong agar ditampilkan
# dari indeks nomor 1 sampai indeks nomor 2
print(web[1:3])
```

```
('Petani Kode', 'https://www.petanikode.com')
```

Mengakses Nilai Tuple

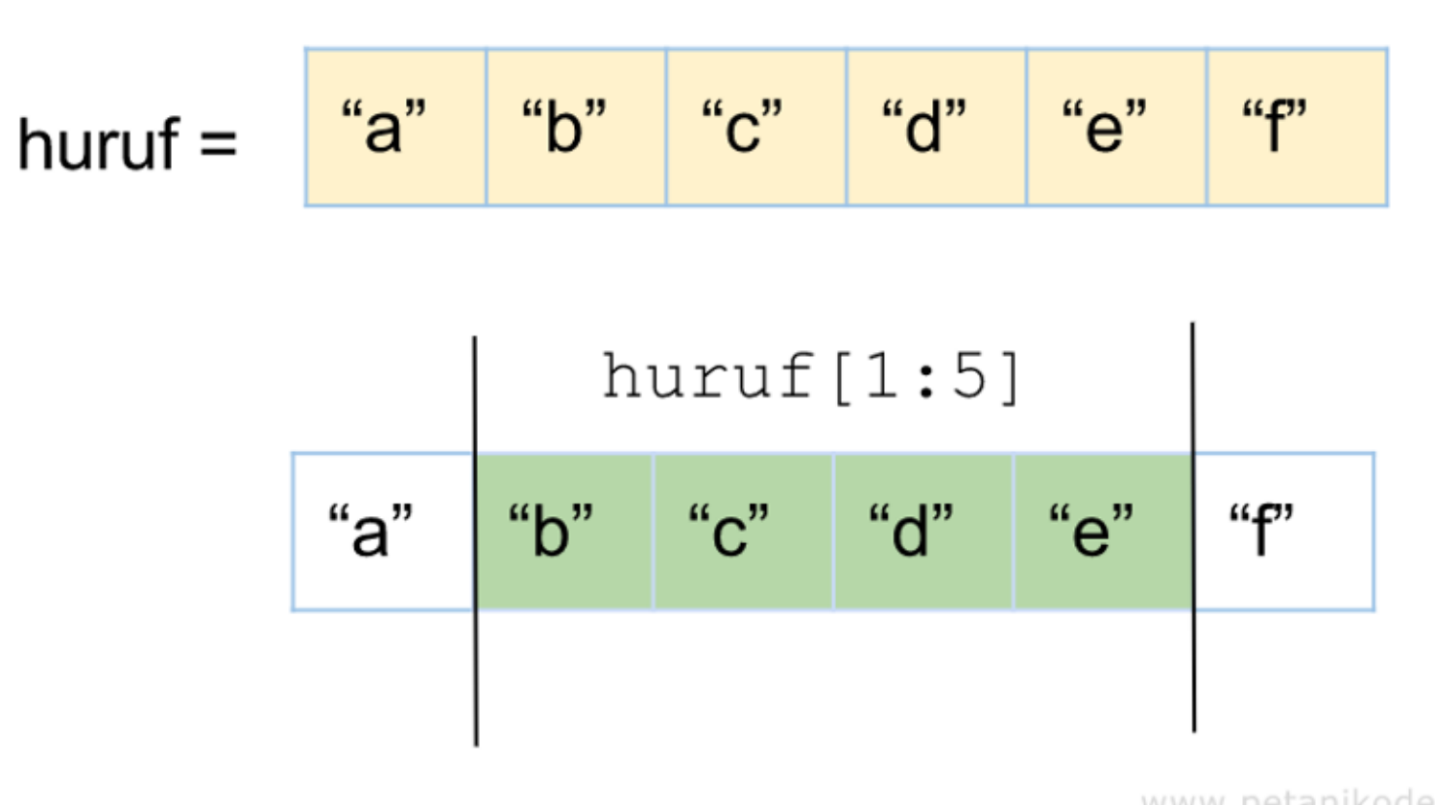
Sama seperti list, Tuple juga memiliki indeks untuk Mengakses item di dalamnya. Indeks Tuple dan list selalu dimulai dari nol 0.

```python
# membuat tuple
nama = ('petani', 'kode', 'linux')

# mengakses nilai tuple
print(nama[1])
```

```
kode
```

Logikanya sama seperti di list.



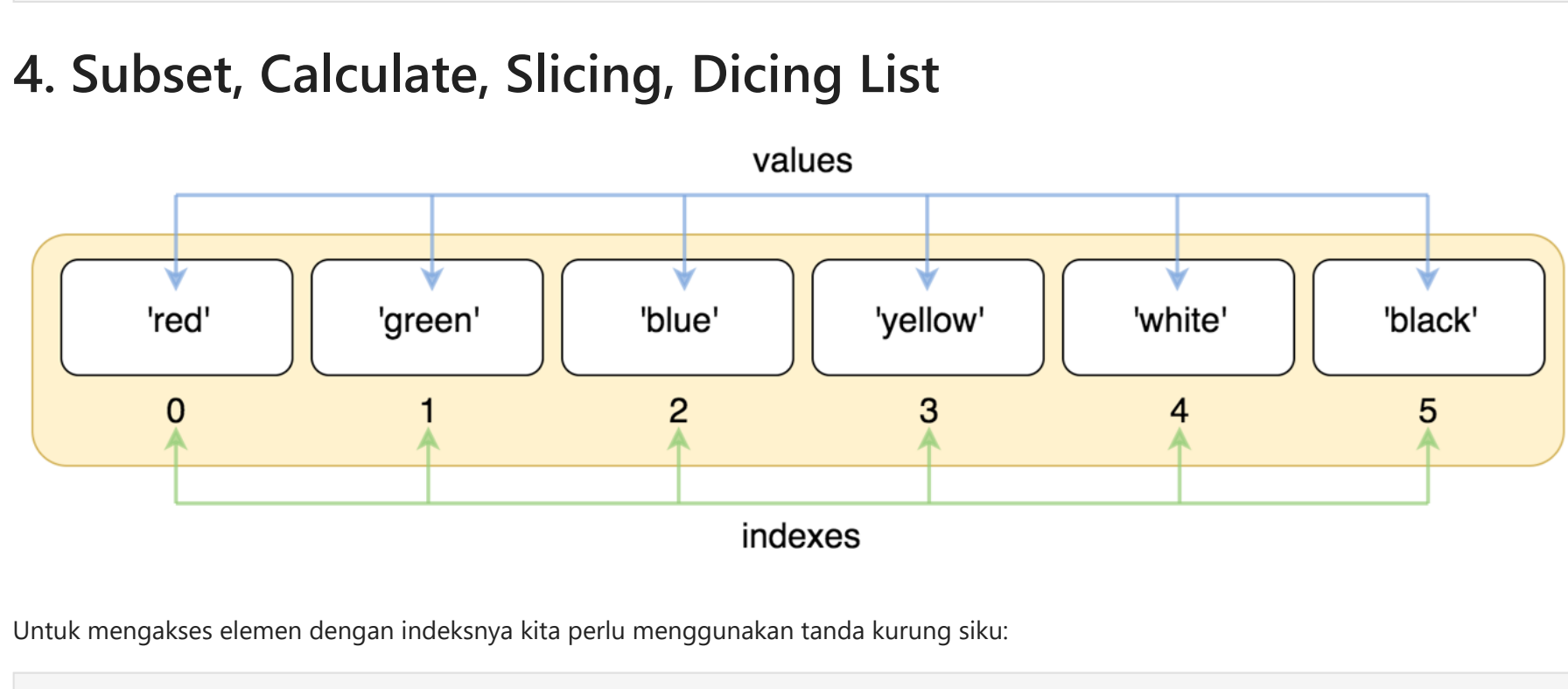
Tuple Nested

Tuple juga bisa nested, artinya Tuple bisa diisi dengan Tuple.

Contoh:

```python
tuple1 = "aku", "cinta", "kamu"
tuple2 = "selama", 3, "tahun"
tuple3 = (tuple1, tuple2) # <- nested tuple
```

4. Subset, Calculate, Slicing, Dicing List



Untuk mengakses elemen dengan indeksnya kita perlu menggunakan tanda kurung siku:

```python
warna = ['merah', 'hijau', 'biru', 'kuning', 'putih', 'hitam']
```

```python
warna[0]
```

```
'merah'
```

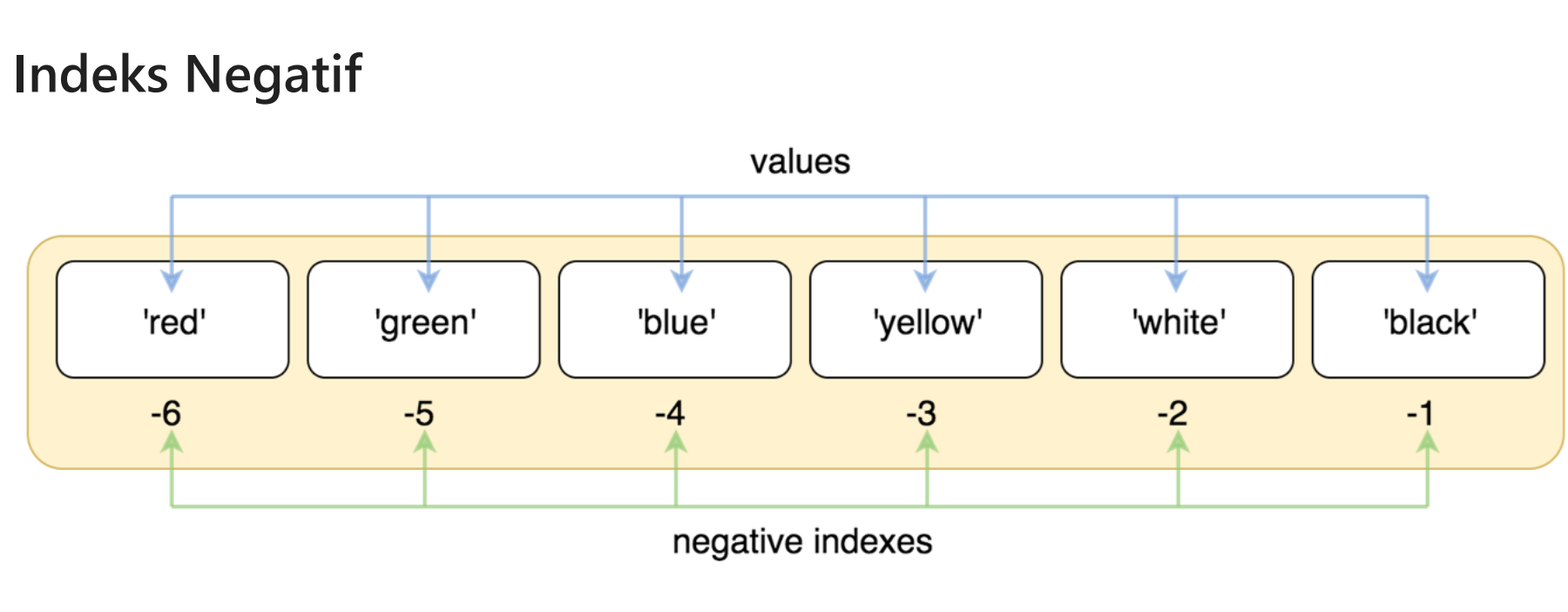
```python
warna[1]
```

```
'hijau'
```

```python
warna[5]
```

```
'hitam'
```

Indeks Negatif



Dalam sistem pengindeksan negatif -1 berhubungan dengan elemen terakhir dari list (nilai 'hitam'), -2 ke belakang (nilai 'putih) dan seterusnya.

```python
warna = ['merah', 'hijau', 'biru', 'kuning', 'putih', 'hitam']
```

```python
warna[-1]
```

```
'hitam'
```

```python
warna[-2]
```

```
'putih'
```

```python
warna[-6]
```

```
'merah'
```

Penggunaan Dasar Irisan

Membuat list dasar:

```python
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
```

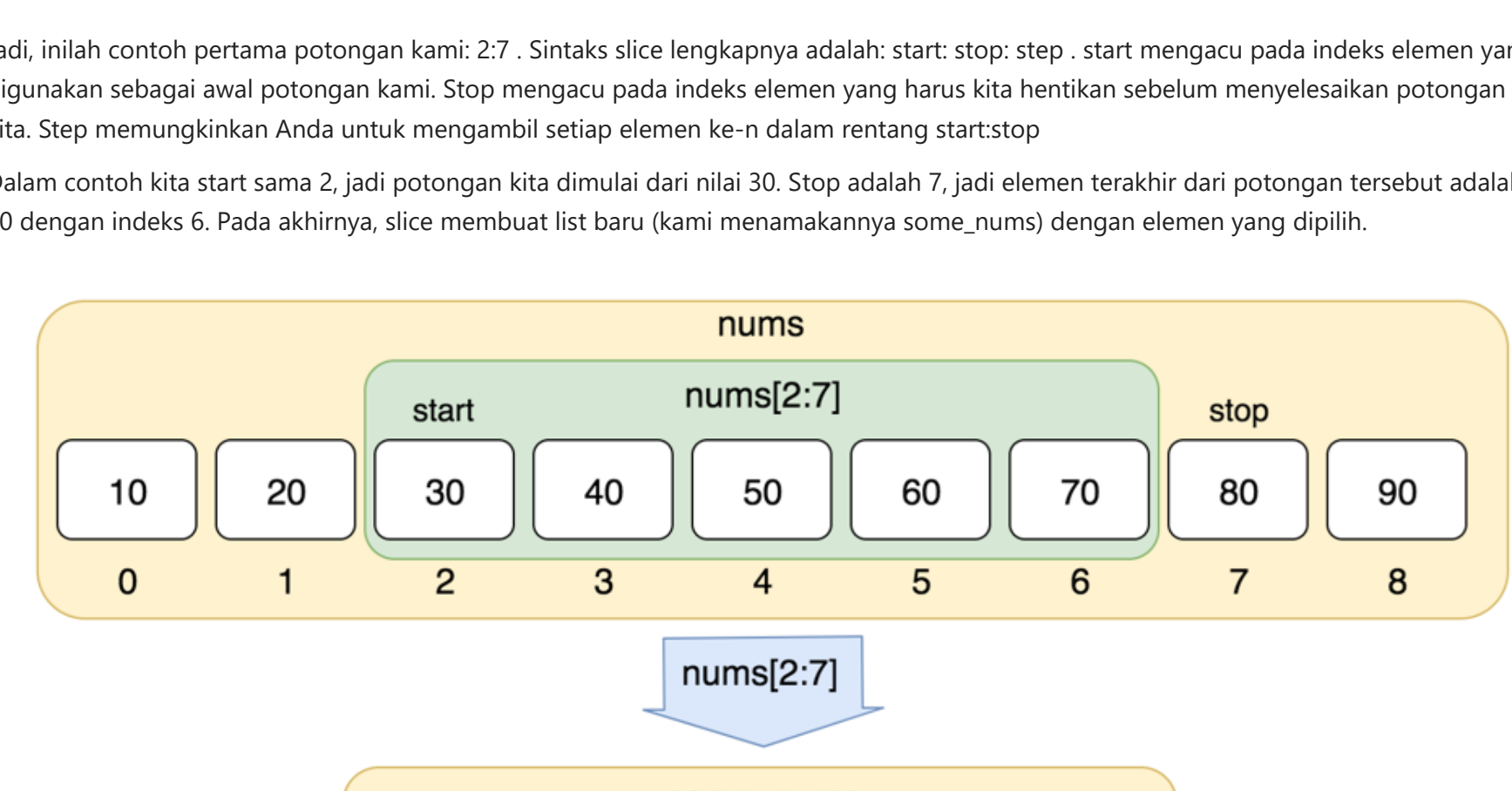
Bagaimana jika kita ingin mengambil sublist dari num list? ini sangat mudah saat menggunakan slice:

```python
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
some_nums = nums[2 : 7]
some_nums
```

```
[30, 40, 50, 60, 70]
```

Jadi, inilah contoh pertama potongan kami: 2:7 . Sintaks slice lengkapnya adalah: start: stop: step . start mengacu pada indeks elemen yang digunakan sebagai awal potongan kami. Stop mengacu pada indeks elemen yang harus kita hentikan sebelum menyelesaikan potongan kita. Step memungkinkan Anda untuk mengambil setiap elemen ke-n dalam rentang start:stop

Dalam contoh kita start sama 2, jadi potongan kita dimulai dari nilai 30. Stop adalah 7, jadi elemen terakhir dari potongan tersebut adalah 70 dengan indeks 6. Pada akhirnya, slice membuat list baru (kami menamakannya some_nums) dengan elemen yang dipilih.



Mengambil N-Elemen Pertama dari List

Notasi slice memungkinkan Anda melewati elemen apa pun dari sintaks lengkap. Jika kita melewatkan start nomornya maka itu dimulai dari 0 indeks:

```python
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
```

```python
nums[ : 5]
```

```
[10, 20, 30, 40, 50]
```

Jadi, angka [: 5] sama dengan angka [0: 5] . Kombinasi ini adalah jalan pintas yang berguna untuk mengambil n elemen pertama dari List.

Mengambil N-Elemen Terakhir dari List

Indeks negatif memungkinkan kita untuk dengan mudah mengambil n-elemen terakhir dari list:

```python
nums = [ 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90 ]
nums[ -3 : ]
```

```
[70, 80, 90]
```

Di sini, stop parameter dilewati. Itu artinya Anda mengambil dari start posisi, sampai akhir List. Kami mulai dari elemen ketiga dari akhir (nilai 70 dengan indeks -3) dan mengambil semuanya sampai akhir. Kami dapat dengan bebas mencampur indeks negatif dan positif dalam start dan stop posisi:

```python
nums = [ 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90 ]
nums[1 : -1]
```

```
[20, 30, 40, 50, 60, 70, 80]
```

```python
nums[-3 : 8]
```

```
[70, 80]
```

```python
nums[ -5 : -1]
```

```
[50, 60, 70, 80]
```

Mengambil Semua Kecuali N Elemen Terakhir Dari list

Penggunaan bagus lainnya dari indeks negatif:

```python
nums = [ 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90 ]
nums  [ : -2 ]
```

```
[10, 20, 30, 40, 50, 60, 70]
```

Mengambil Setiap Elemen Ke N dari List

Bagaimana jika kita hanya ingin memiliki setiap elemen 2-nd nums? Di sinilah step parameter berperan:

```python
nums = [10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90]
nums[ :: 2 ]
```

```
[10, 30, 50, 70, 90]
```

Di sini kami menghilangkan start/ stop parameter dan hanya menggunakan step. Dengan menyediakan start kita dapat melewati beberapa elemen:

```python
nums[1 :: 2]
```

```
[20, 40, 60, 80]
```

Dan jika kita tidak ingin memasukkan beberapa elemen pada akhirnya, kita juga bisa menambahkan stop parameter:

```python
nums[1 : -3 : 2]
```

```
[20, 40, 60]
```

Menggunakan Langkah Negatif dan List Terbaik

Kita dapat menggunakan negatif step untuk mendapatkan List terbalik:

```python
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
nums[-2 : : -1]
```

```
[80, 70, 60, 50, 40, 30, 20, 10]
```

Negatif step mengubah cara, notasi irisan berfungsi. Itu membuat slice dibangun dari List terbelakang. Jadi, ini berpindah dari elemen terakhir ke elemen pertama. Itu sebabnya kami mendapatkan List terbalik dengan langkah negatif.

Karena keanehan ini, start dan stop harus disediakan dari kanan ke kiri juga. Misalnya, jika Anda ingin memiliki List terbalik yang dimulai dari 80:

```python
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
nums[-2 : 1 : -1]
```

```
[80, 70, 60, 50, 40, 30]
```

Jadi, kita mulai dari -2 elemen (nilai 80) dan pergi dari kanan ke kiri mengumpulkan semua elemen dalam List terbalik.

Objek Slice

Tetapi bagaimana jika kita ingin menggunakan potongan yang sama berulang kali. Apakah ada cara untuk membuat objek irisan daripada hanya menggunakan bentuk sintaksis?

Ini dapat dilakukan dengan menggunakan slice fungsi:

```python
five_items_after_second = slice(2, 2 + 5)
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
colors = ['red', 'green', 'blue', 'yellow', 'white', 'black', 'silver']
```

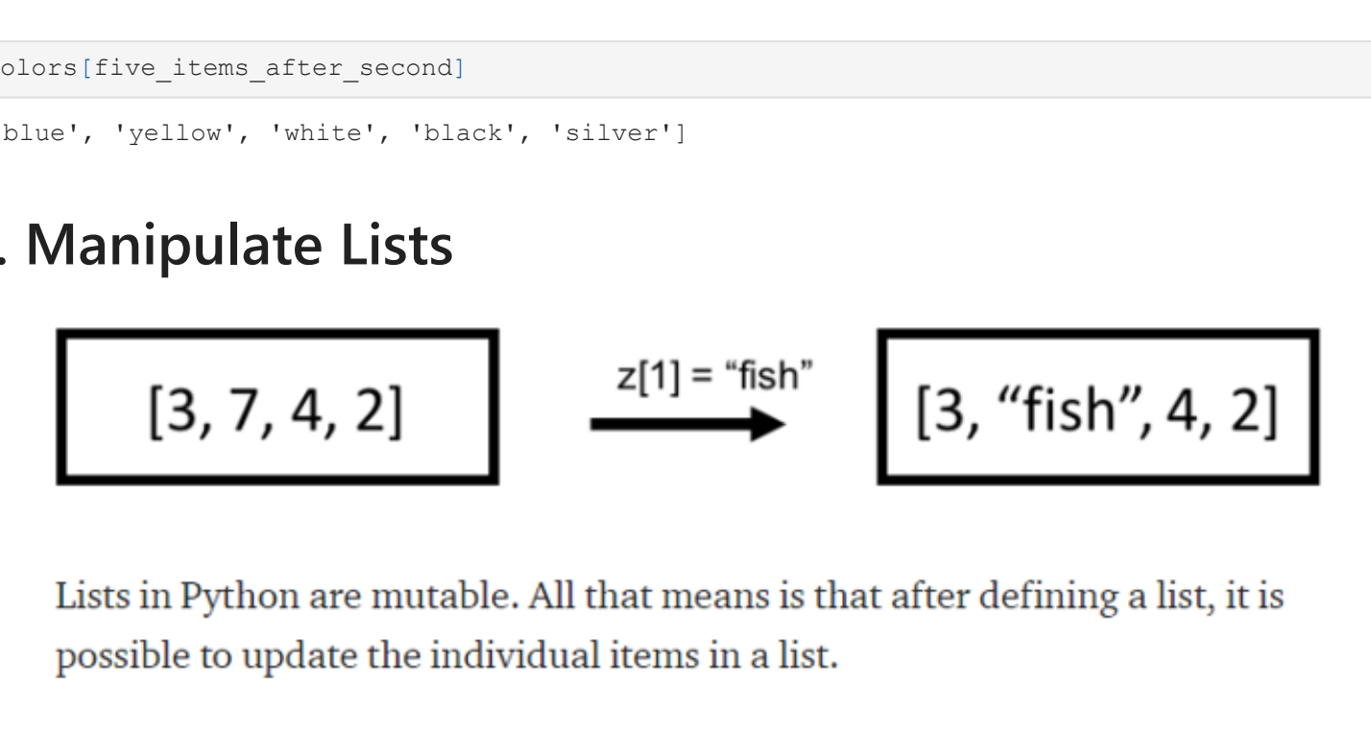
```python
nums[five_items_after_second]
```

```
[30, 40, 50, 60, 70]
```

```python
colors[five_items_after_second]
```

```
['blue', 'yellow', 'white', 'black', 'silver']
```

5. Manipulate Lists



Lists in Python are mutable. All that means is that after defining a list, it is possible to update the individual items in a list.

```python
# Defining a list
z = [3, 7, 4, 2]

# Update the item at index 1 with the string "fish"
z[1] = "fish"
print(z)
```

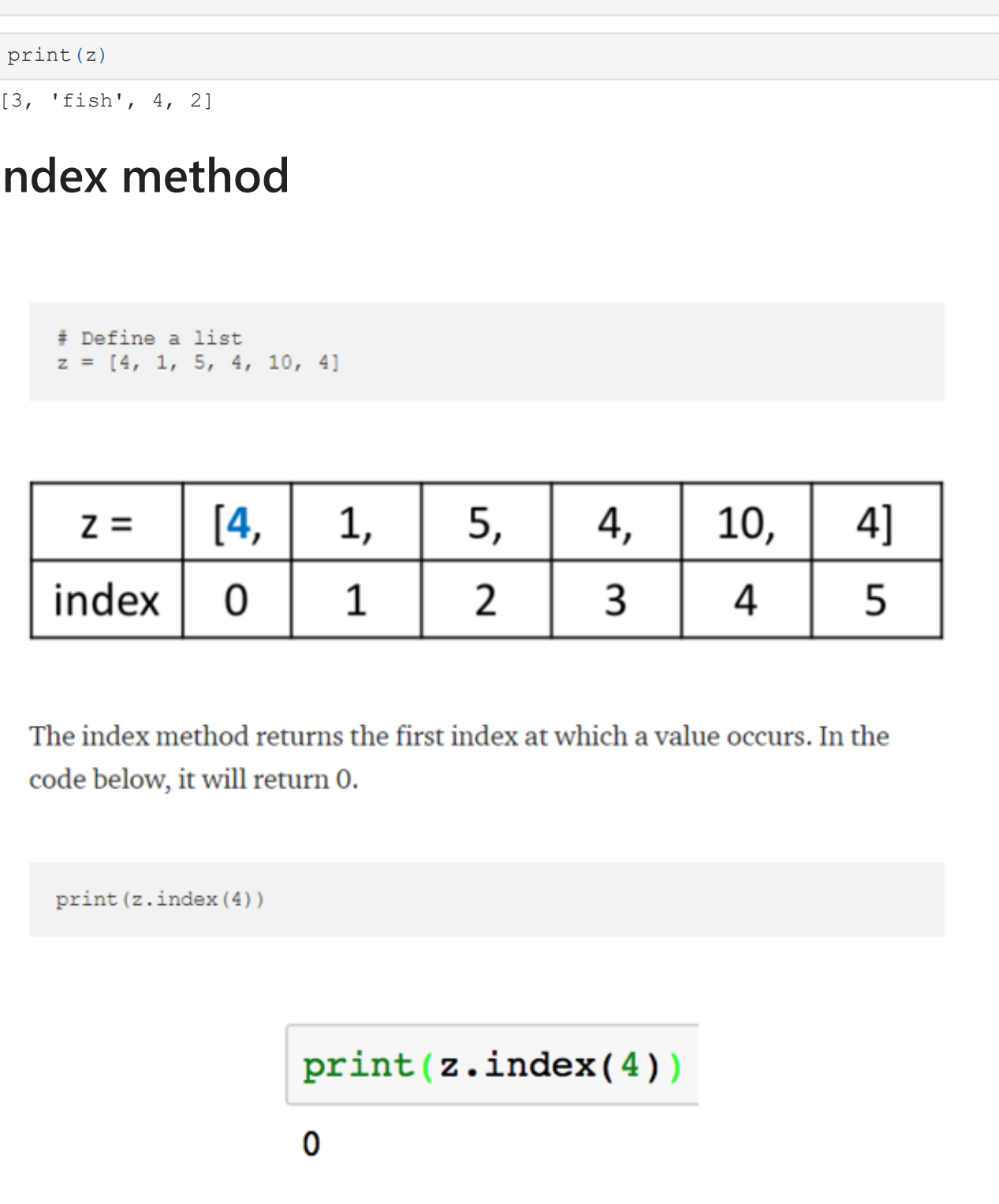
```python
z = [3, 7, 4, 2]
```

```python
# mengganti nilai pada indeks 1
z[1] = "fish"
```

```python
print(z)
```

```
[3, 'fish', 4, 2]
```

index method



```python
# Define a list
z = [4, 1, 5, 4, 10, 4]
```

z =	[4,	1,	5,	4,	10,	4]
index	0	1	2	3	4	5

The index method returns the first index at which a value occurs. In the code below, it will return 0.

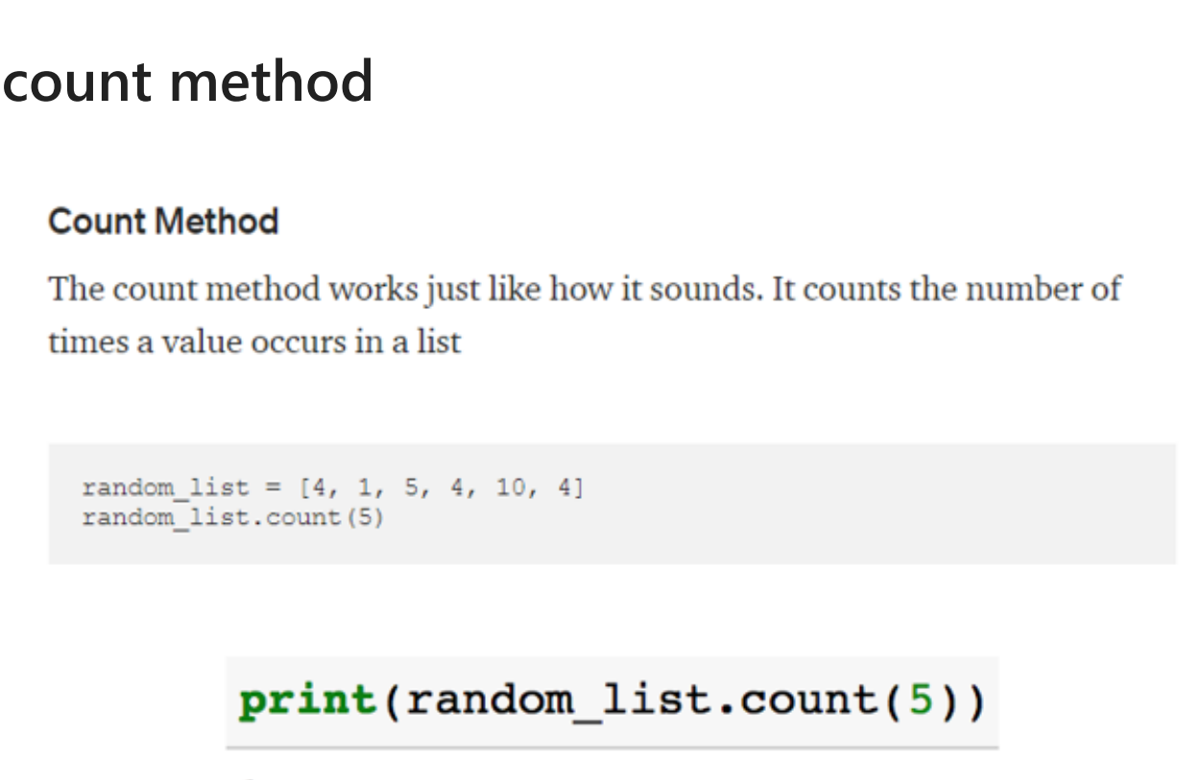
```python
print(z.index(4))
```

```python
print(z.index(4))
```

```
0
```

z =	[4,	1,	5,	4,	10,	4]
index	0	1	2	3	4	5

count method



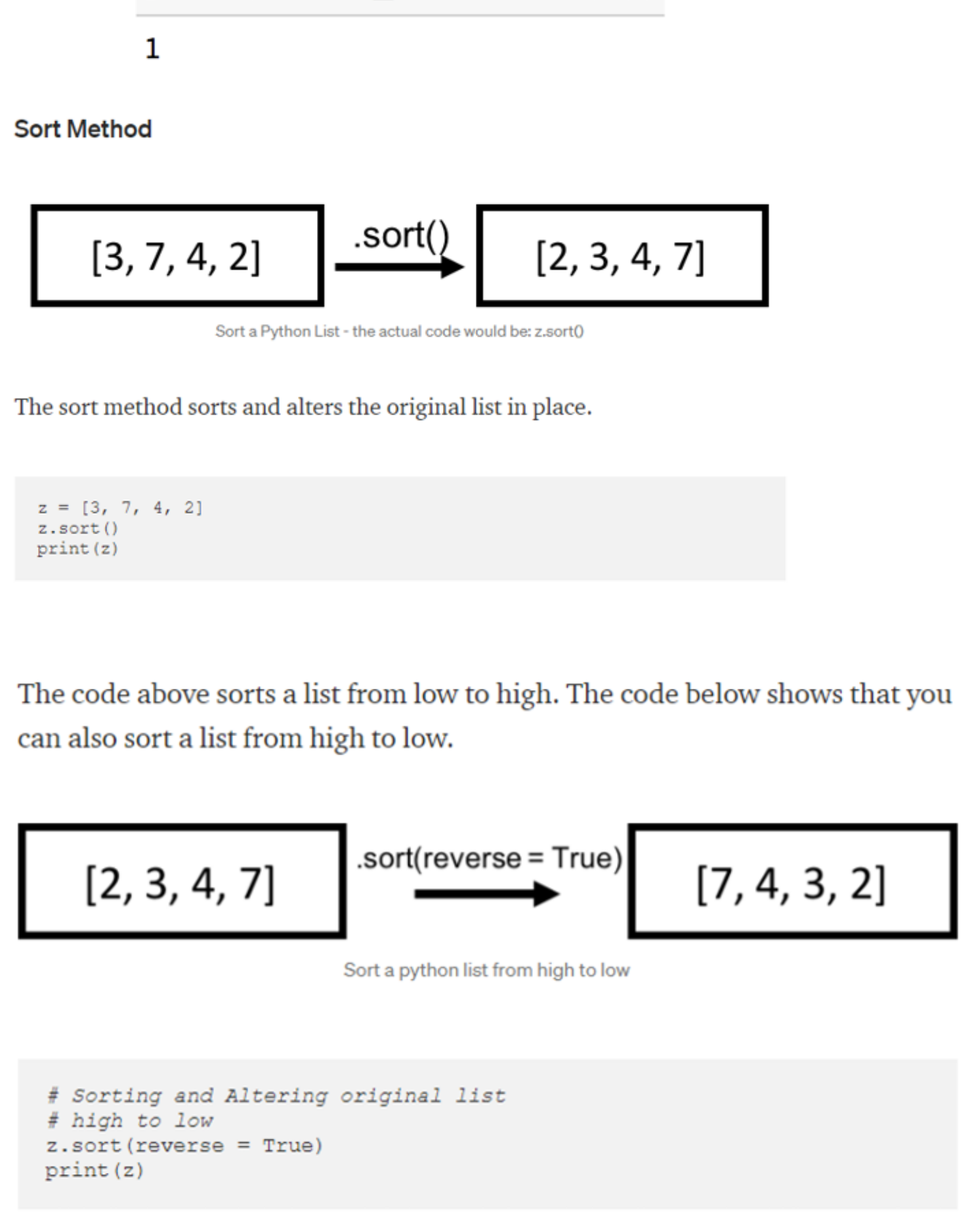
Count Method

The count method works just like how it sounds. It counts the number of times a value occurs in a list

```python
random_list = [4, 1, 5, 4, 10, 4]
random_list.count(5)
```

```python
print(random_list.count(5))
```

```
1
```



Sort Method

Sort a Python List - the actual code would be: z.sort()

The sort method sorts and alters the original list in place.

```python
z = [3, 7, 4, 2]
z.sort()
print(z)
```

The code above sorts a list from low to high. The code below shows that you can also sort a list from high to low.

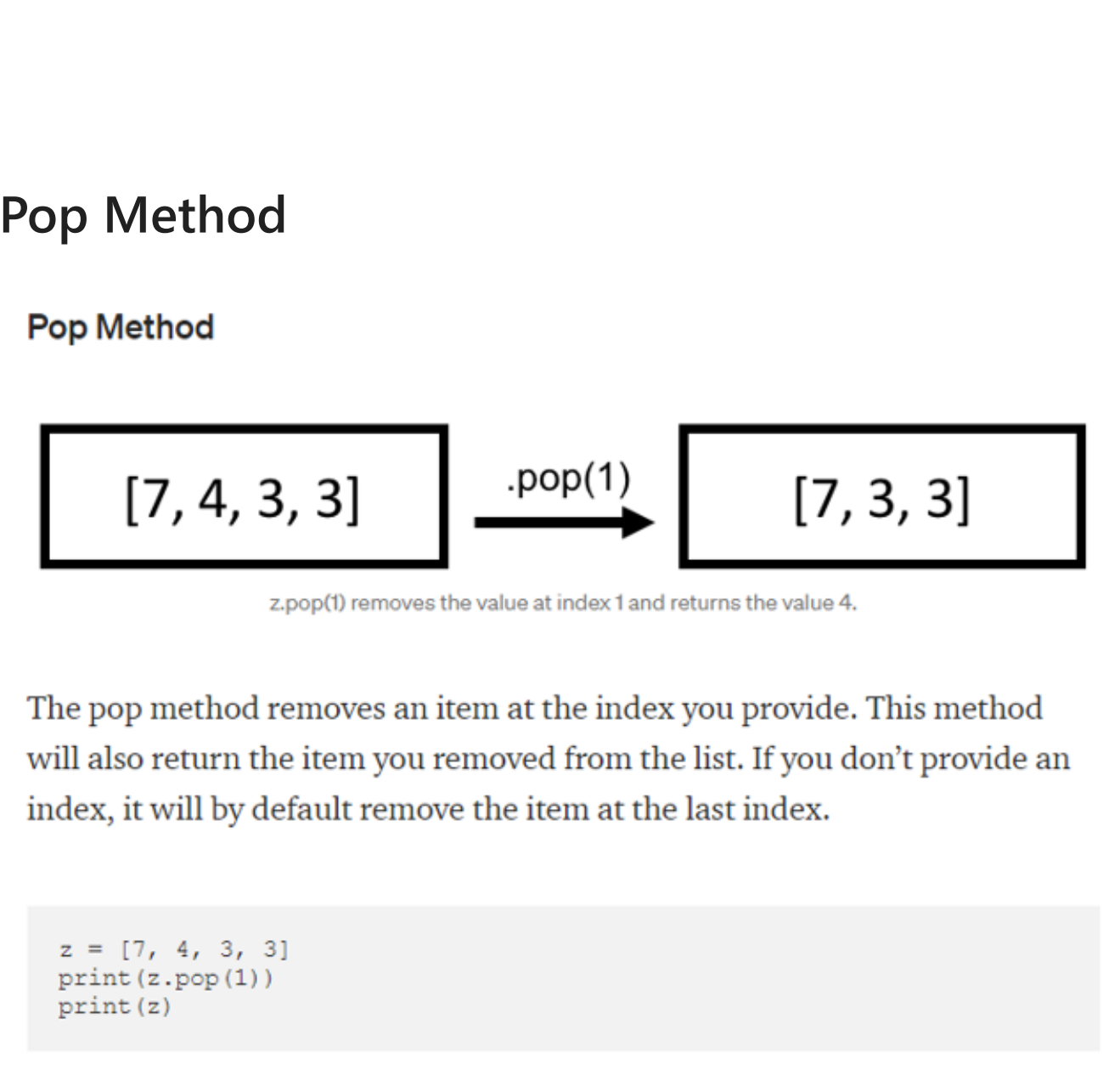
Sort a python list from high to low

```python
# Sorting and Altering original list
# high to low
z.sort(reverse = True)
print(z)
```

```python
z.sort(reverse = True)
print(z)
```

```
[7, 4, 3, 2]
```

Pop Method



Pop Method

z.pop(1) removes the value at index 1 and returns the value 4.

The pop method removes an item at the index you provide. This method will also return the item you removed from the list. If you don't provide an index, it will by default remove the item at the last index.

```python
z = [7, 4, 3, 3]
print(z.pop(1))
print(z)
```

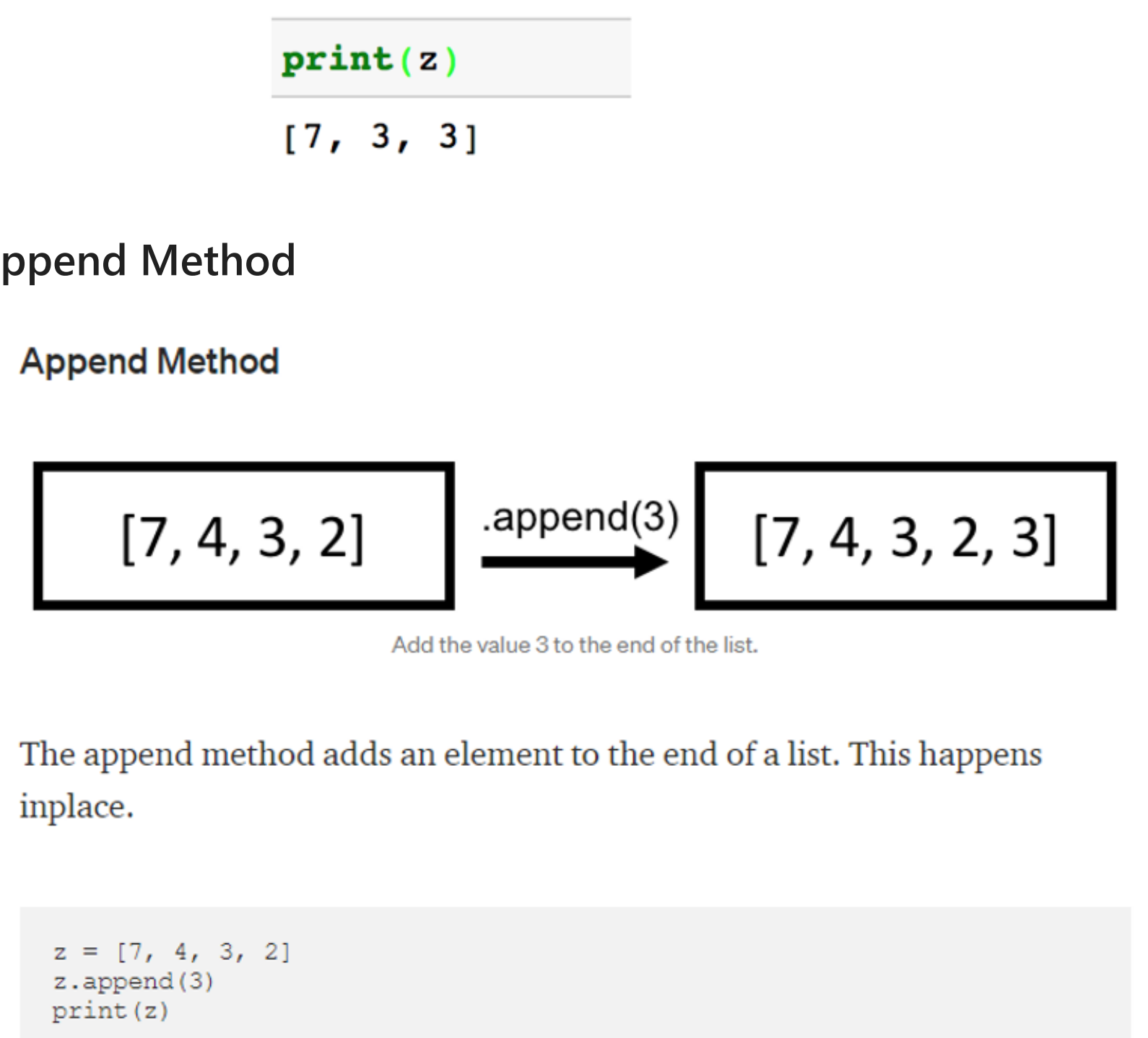
```python
print(z.pop(1))
```

```
4
```

```python
print(z)
```

```
[7, 3, 3]
```

Append Method



Append Method

Add the value 3 to the end of the list.

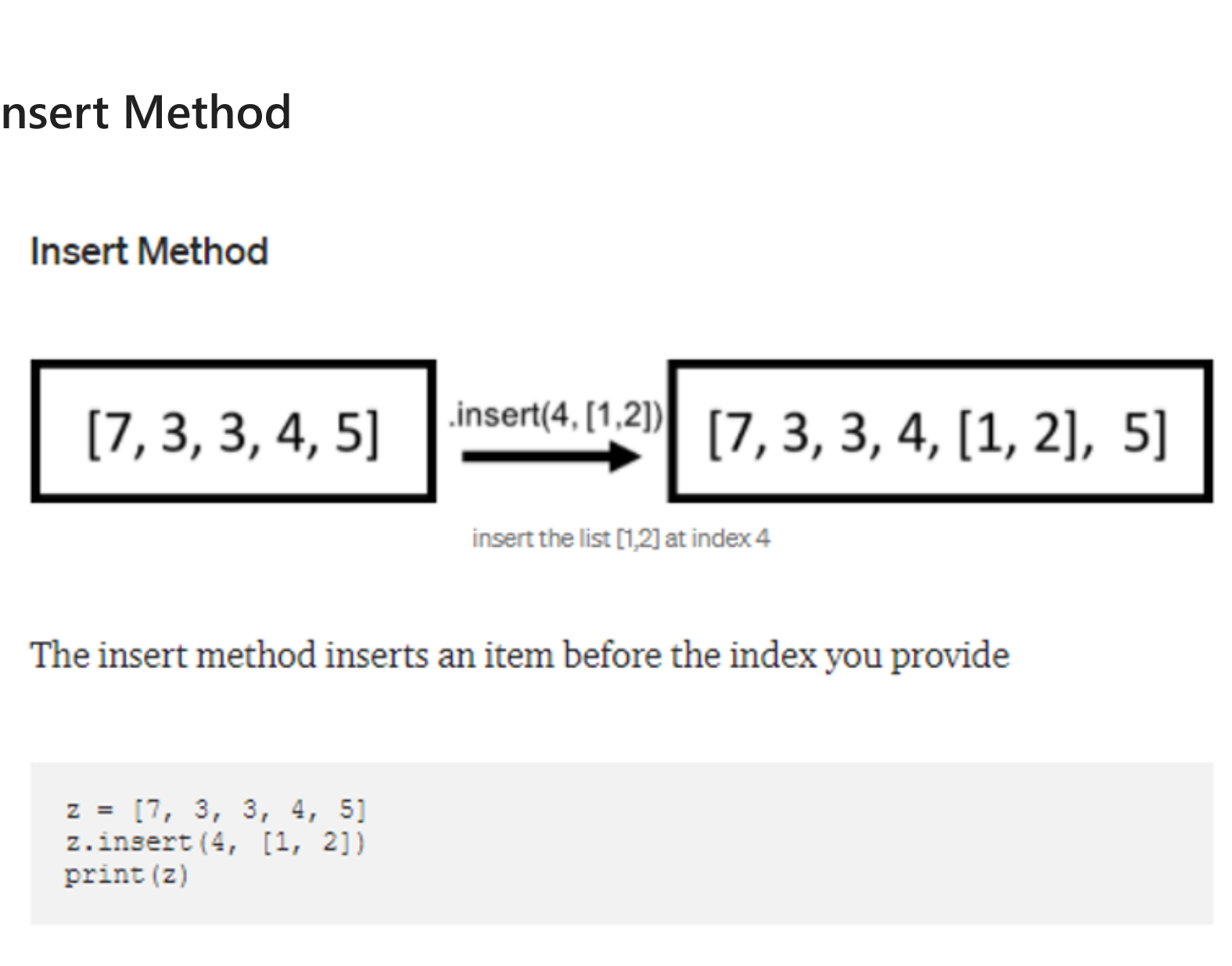
The append method adds an element to the end of a list. This happens inplace.

```python
z = [7, 4, 3, 2]
z.append(3)
print(z)
```

```python
z.append(3)
print(z)
```

```
[7, 4, 3, 2, 3]
```

Insert Method



Insert Method

insert the list [1,2] at index 4

The insert method inserts an item before the index you provide

```python
z = [7, 3, 3, 4, 5]
z.insert(4, [1, 2])
print(z)
```

```python
z.insert(4, [1, 2])
print(z)
```

```
[7, 3, 3, 4, [1, 2], 5]
```

6. Replace List Elements

Ganti Bagian dari List

Penetapan slice memungkinkan Anda memperbarui bagian dari list dengan nilai baru:

```python
nums = [10, 20, 30, 40, 50, 60, 70, 80, 90]
```

```python
nums[:4] = [1, 2, 3, 4]
nums
```

```
[1, 2, 3, 4, 50, 60, 70, 80, 90]
```

Di sini kami tidak mengubah jumlah elemen dalam list. Hanya beberapa nilai list yang diperbarui.

Ganti dan Ubah Ukuran Bagian dari list

Kita dapat mengganti bagian dari list dengan potongan yang lebih besar:

In [45]:
```
nums = [10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90]
```

In [46]:
```
nums[ : 4] = [ 1 , 2 , 3 , 4 , 5 , 6 , 7]
```

In [47]:
```
nums
```

Out[47]:
```
[1, 2, 3, 4, 5, 6, 7, 50, 60, 70, 80, 90]
```

Dalam hal ini kami memperluas list asli.

Anda juga dapat mengganti bagian yang lebih besar dengan jumlah barang yang lebih sedikit:

In [48]:
```
nums = [ 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90 ]
```

In [49]:
```
nums[ : 4] = [1]
```

In [50]:
```
nums
```

Out[50]:
```
[1, 50, 60, 70, 80, 90]
```

Ganti Setiap Elemen ke N

Menambahkan step memungkinkan untuk mengganti setiap elemen ke-n dengan nilai baru:

In [51]:
```
nums = [10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90]
```

In [52]:
```
nums[ :: 2] = [1 , 1 , 1 , 1 , 1]
nums
```

Out[52]:
```
[1, 20, 1, 40, 1, 60, 1, 80, 1]
```

Kita juga dapat menggunakan negatif step :

In [53]:
```
nums = [ 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90 ]
```

In [54]:
```
nums[ :: -2 ] = [ 1 , 2 , 3 , 4 , 5]
nums
```

Out[54]:
```
[5, 20, 4, 40, 3, 60, 2, 80, 1]
```

Dengan memberikan start dan stop nilai, kita dapat mempersempit area penggantian:

In [55]:
```
nums = [ 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90 ]
```

In [56]:
```
nums [ 1 : 5 : 2 ] = [ 2 , 4 ]
nums
```

Out[56]:
```
[10, 2, 30, 4, 50, 60, 70, 80, 90]
```

7. Extend and Delete List

Hapus List

Kami juga dapat dengan mudah menghapus elemen apa pun dari List dengan menggunakan pengindeksan dan del pernyataan:

In [57]:
```
keranjang = [ 'roti' , 'mentega' , 'susu' ]
```

In [58]:
```
del(keranjang[0])
keranjang
```

Out[58]:
```
['mentega', 'susu']
```

In [59]:
```
# menghapus susu atau indeks 1
del(keranjang[1])
```

In [60]:
```
keranjang
```

Out[60]:
```
['mentega']
```

Kami juga dapat menggunakan del pernyataan untuk menghapus potongan dari list:

In [61]:
```
nums = [ 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90 ]
```

In [62]:
```
del(nums[3 : 7])
```

In [63]:
```
nums
```

Out[63]:
```
[10, 20, 30, 80, 90]
```

Di sini kami telah menghapus banyak elemen di tengah nums list.

Kami juga dapat memberikan step parameter untuk memotong dan menghapus setiap elemen ke-n:

In [64]:
```
nums = [ 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90 ]
```

In [65]:
```
del(nums[::2])
```

In [66]:
```
nums
```

Out[66]:
```
[20, 40, 60, 80]
```

Dengan sintaks lengkap, kita dapat mengatur batasan untuk elemen yang akan dihapus:

In [67]:
```
nums = [ 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90]
```

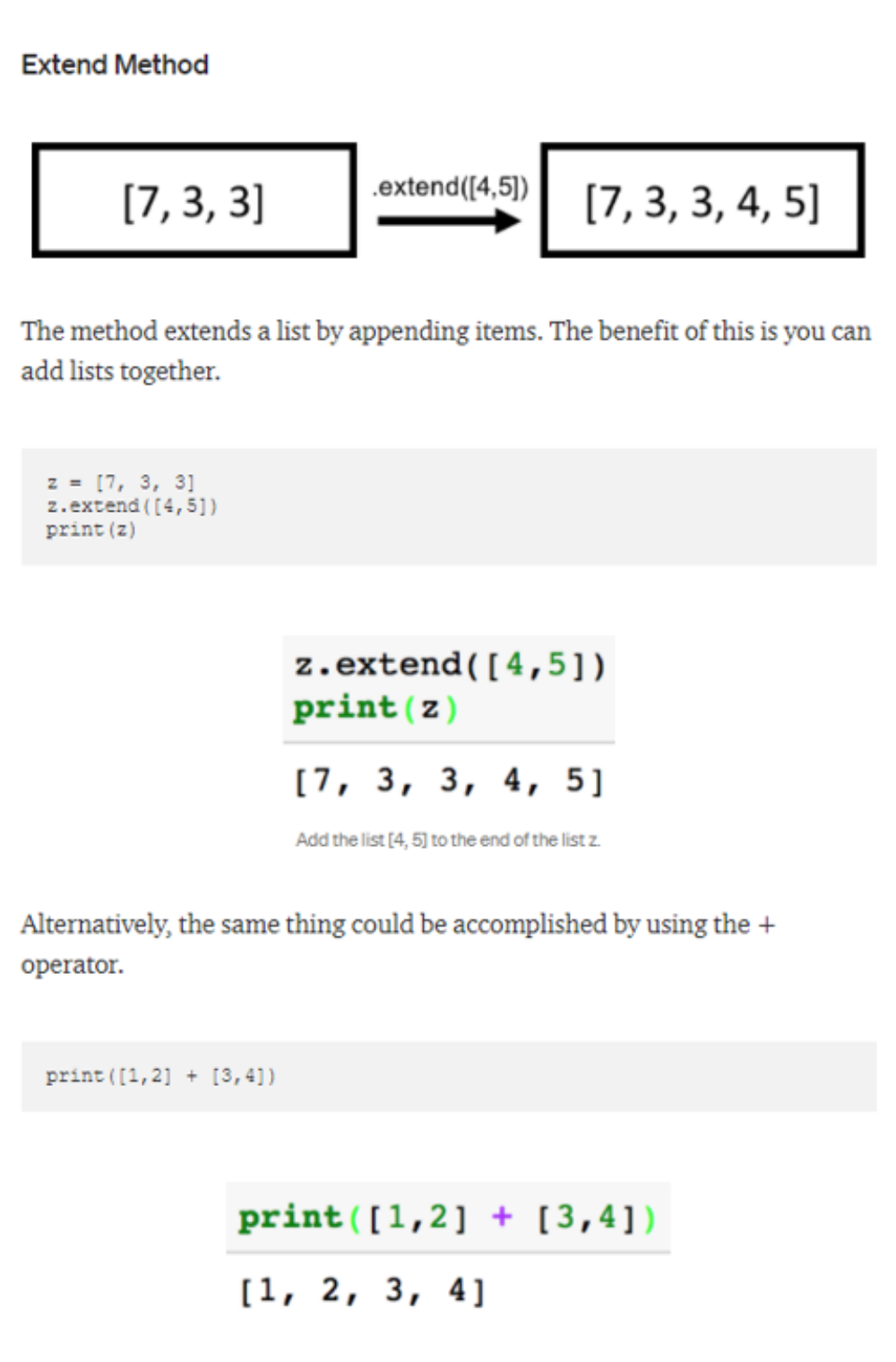
In [68]:
```
del(nums[ 1 : 7 : 2 ])
```

In [69]:
```
nums
```

Out[69]:
```
[10, 30, 50, 70, 80, 90]
```

Jadi, kami mulai menghapus dari 20 (indeks 1) dan menghapus setiap elemen 2-nd sampai nilai 80 (indeks 7).

Extend Method


Extend Method

The method extends a list by appending items. The benefit of this is you can add lists together.

```
z = [7, 3, 3]
z.extend([4,5])
print(z)
```

```
z.extend([4,5])
print(z)
```
```
[7, 3, 3, 4, 5]
```

Add the list [4, 5] to the end of the list z.

Alternatively, the same thing could be accomplished by using the + operator.

```
print([1,2] + [3,4])
```

```
print([1,2] + [3,4])
```
```
[1, 2, 3, 4]
```