
TUPLES

Sebuah tupel adalah urutan objek Python yang tidak berubah. Tupel adalah urutan, seperti daftar. Perbedaan antara tupel dan daftar adalah, tupel tidak dapat diubah tidak seperti daftar dan tupel menggunakan tanda kurung, sedangkan daftar menggunakan tanda kurung siku.

Membuat tuple semudah memasukkan nilai-nilai yang dipisahkan koma. Opsional Anda dapat memasukkan nilai-nilai yang dipisahkan koma ini di antara tanda kurung juga. Misalnya -

```
Tup1 = ('fisika', 'kimia', 1997, 2000);
```

```
Tup2 = (1, 2, 3, 4, 5);
```

```
Tup3 = "a", "b", "c", "d";
```

Tuple kosong ditulis sebagai dua tanda kurung yang tidak berisi apa -

```
tup1 = ();
```

Untuk menulis tupel yang berisi satu nilai, Anda harus menyertakan koma, meskipun hanya ada satu nilai -

```
Tup1 = (50,);
```

Seperti indeks string, indeks tuple mulai dari 0, dan mereka dapat diiris, digabungkan, dan seterusnya.

Mengakses Nilai pada Tuples:

Untuk mengakses nilai dalam tupel, gunakan tanda kurung siku untuk mengiris beserta indeks atau indeks untuk mendapatkan nilai yang tersedia pada indeks tersebut. Misalnya -

```
#!/usr/bin/python
```

```
Tup1 = ('fisika', 'kimia', 1997, 2000);
```

```
Tup2 = (1, 2, 3, 4, 5, 6, 7);
```

```
Cetak "tup1 [0]:", tup1 [0]
```

Cetak "tup2 [1: 5]:", tup2 [1: 5]

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

tup1 [0]: fisika

Tup2 [1: 5]: [2, 3, 4, 5]

Memperbarui Tupel

Tupel tidak berubah yang berarti Anda tidak dapat memperbarui atau mengubah nilai elemen tupel. Anda dapat mengambil bagian dari tupel yang ada untuk membuat tupel baru seperti ditunjukkan oleh contoh berikut -

```
#!/usr/bin/python
```

```
Tup1 = (12, 34.56);
```

```
Tup2 = ('abc', 'xyz');
```

```
# Tindakan berikut tidak berlaku untuk tupel
```

```
# Tup1 [0] = 100;
```

```
# Jadi mari kita buat tupel baru sebagai berikut
```

```
Tup3 = tup1 + tup2;
```

Cetak tup3

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
(12, 34.56, 'abc', 'xyz')
```

Hapus Elemen Tuple

Menghapus elemen tuple individual tidak mungkin dilakukan. Tentu saja, tidak ada yang salah dengan menggabungkan tuple lain dengan unsur-unsur yang tidak diinginkan dibuang.

Untuk secara eksplisit menghapus keseluruhan tuple, cukup gunakan del statement. Sebagai contoh:

```
#!/usr/bin/python
```

```
Tup = ('fisika', 'kimia', 1997, 2000);
```

```
Cetak tup
```

```
Del tup;
```

```
Cetak "Setelah menghapus tup:"
```

```
Cetak tup
```

Ini menghasilkan hasil berikut. Perhatikan pengecualian yang diangkat, ini karena setelah del tup tuple tidak ada lagi -

```
('Fisika', 'kimia', 1997, 2000)
```

```
Setelah menghapus tup:
```

```
Traceback (panggilan terakhir):
```

```
File "test.py", baris 9, di <module>
```

```
    Cetak tup;
```

```
NameError: nama 'tup' tidak didefinisikan
```

Operasi Tuple Dasar

Tuple merespons operator + dan * seperti string; Mereka berarti penggabungan dan pengulangan di sini juga, kecuali hasilnya adalah tuple baru, bukan string.

Sebenarnya, tuple menangani semua operasi urutan umum yang kami gunakan pada senar di bab sebelumnya -

Python Expression	Results	Description
len((1, 2, 3))	3	Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!') * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

Indexing, Slicing, dan Matrixes

Karena tuple adalah urutan, pengindeksan dan pengiris bekerja dengan cara yang sama untuk tuple seperti yang mereka lakukan untuk string. Dengan asumsi masukan berikut -
`L = ('spam', 'Spam', 'SPAM!')`

Python Expression	Results	Description
<code>L[2]</code>	<code>'SPAM!'</code>	Offsets start at zero
<code>L[-2]</code>	<code>'Spam'</code>	Negative: count from the right
<code>L[1:]</code>	<code>['Spam', 'SPAM!']</code>	Slicing fetches sections

Tidak melampirkan delimiters

Setiap kumpulan beberapa objek, yang dipisahkan koma, ditulis tanpa mengidentifikasi simbol, yaitu tanda kurung untuk daftar, tanda kurung untuk tuple, dll., Default tuple, seperti yang ditunjukkan dalam contoh singkat ini -

```
#!/usr/bin/python
```

```
cetak 'abc', -4.24e93, 18 + 6.6j, 'xyz'
```

```
x, y = 1, 2;
```

```
Cetak "Nilai x, y:", x, y
```

Bila kode diatas dieksekusi, maka menghasilkan hasil sebagai berikut -

```
abc -4.24e + 93 (18 + 6.6j) xyz
```

```
Nilai x, y: 1 2
```

Built-in Fungsi Tuple

Python mencakup fungsi tuple berikut –

SN	Function with Description
1	<code>cmp(tuple1, tuple2)</code> Compares elements of both tuples.
2	<code>len(tuple)</code> Gives the total length of the tuple.
3	<code>max(tuple)</code> Returns item from the tuple with max value.
4	<code>min(tuple)</code> Returns item from the tuple with min value.
5	<code>tuple(seq)</code> Converts a list into tuple.

Dalam pemrograman Python, tuple mirip dengan daftar. Perbedaan antara keduanya adalah kita tidak bisa mengubah unsur tuple begitu diberikan sedangkan dalam daftar, elemen bisa diubah.

Keuntungan Tuple over List

Karena, tupel sangat mirip dengan daftar, keduanya juga digunakan dalam situasi yang sama.

Namun, ada beberapa keuntungan dari penerapan tupel dari daftar. Di bawah ini tercantum beberapa keuntungan utama:

Kami umumnya menggunakan tuple untuk tipe data heterogen dan berbeda untuk tipe data homogen (sejenis).

Karena tupel tidak dapat diubah, iterasi melalui tupel lebih cepat daripada daftar. Jadi ada sedikit peningkatan kinerja.

Tupel yang mengandung unsur yang tidak berubah dapat digunakan sebagai kunci untuk kamus. Dengan daftar, ini tidak mungkin.

Jika Anda memiliki data yang tidak berubah, menerapkannya sebagai tupel akan menjamin bahwa itu tetap dilindungi penulisan.

Dalam pemrograman Python, tuple mirip dengan daftar. Perbedaan antara keduanya adalah kita tidak bisa mengubah unsur tuple begitu diberikan sedangkan dalam daftar, elemen bisa diubah.

Keuntungan Tuple over List

Karena, tupel sangat mirip dengan daftar, keduanya juga digunakan dalam situasi yang sama.

Namun, ada beberapa keuntungan dari penerapan tupel dari daftar. Di bawah ini tercantum beberapa keuntungan utama:

Kami umumnya menggunakan tuple untuk tipe data heterogen dan berbeda untuk tipe data homogen (sejenis).

Karena tupel tidak dapat diubah, iterasi melalui tupel lebih cepat daripada

daftar. Jadi ada sedikit peningkatan kinerja.

Tupel yang mengandung unsur yang tidak berubah dapat digunakan sebagai kunci kamus. Dengan daftar, ini tidak mungkin.

Jika Anda memiliki data yang tidak berubah, menerapkannya sebagai tupel akan menjamin bahwa itu tetap dilindungi penulisan.

Membuat Tuple

Sebuah tuple dibuat dengan menempatkan semua item (elemen) di dalam tanda kurung (), dipisahkan dengan koma. Tanda kurung bersifat opsional namun merupakan praktik yang baik untuk menuliskannya.

Sebuah tuple dapat memiliki sejumlah item dan mereka mungkin memiliki tipe yang berbeda (integer, float, list, string etc.).

```
# empty tuple
```

```
# Output: ()
```

```
my _tuple = ()
```

```
print(my _tuple)
```

```
# tuple having integers
```

```
# Output: (1, 2, 3)
```

```
my _tuple = (1, 2, 3)
```

```
print(my _tuple)
```

```
# tuple with mixed datatypes
```

```
# Output: (1, "Hello", 3.4)
```

```
my _tuple = (1, "Hello", 3.4)
```

```
print(my _tuple)
```

```
# nested tuple
```

```
# Output: ("mouse", [8, 4, 6], (1, 2, 3))
```

```
my _tuple = ("mouse", [8, 4, 6], (1, 2, 3))
```

```
print(my _tuple)
```

```
# tuple can be created without parentheses
```

```
# also called tuple packing
```

```
# Output: 3, 4.6, "dog"
```

```
my _tuple = 3, 4.6, "dog"
```

```
print(my _tuple)
```

```
# tuple unpacking is also possible
```

```
# Output:
```

```
# 3
```

```
# 4.6
```

```
# dog
```

```
a, b, c = my _tuple
```

```
print(a)
```

```
print(b)
```

```
print(c)
```

Membuat tuple dengan satu elemen agak rumit.

Memiliki satu elemen dalam kurung saja tidak cukup. Kita membutuhkan

koma trailing untuk menunjukkan bahwa sebenarnya ada tuple.

```
# only parentheses is not enough
```

```
# Output: <class 'str'>
```

```
my_tuple = ("hello")
```

```
print(type(my_tuple))
```

```
# need a comma at the end
```

```
# Output: <class 'tuple'>
```

```
my_tuple = ("hello",)
```

```
print(type(my_tuple))
```

```
# parentheses is optional
```

```
# Output: <class 'tuple'>
```

```
my_tuple = "hello",
```

```
print(type(my_tuple))
```

Mengakses Elemen dalam Tuple

Ada berbagai cara untuk mengakses elemen tuple.

1. Pengindeksan

Kita bisa menggunakan operator indeks `[]` untuk mengakses item di tuple dimana indeks dimulai dari 0.

Jadi, tuple yang memiliki 6 elemen akan memiliki indeks dari 0 sampai

5. Mencoba mengakses elemen lain yang (6, 7, ...) akan menghasilkan `IndexError`.

Indeks harus berupa bilangan bulat, jadi kita tidak bisa menggunakan float atau jenis lainnya. Ini akan menghasilkan TypeError.

Demikian juga, tuple bersarang diakses menggunakan pengindeksan nested, seperti yang ditunjukkan pada contoh di bawah ini.

```
my_tuple = ('p','e','r','m','i','t')
```

```
# Output: 'p'
```

```
print(my_tuple[0])
```

```
# Output: 't'
```

```
print(my_tuple[5])
```

```
# index must be in range
```

```
# If you uncomment line 14,
```

```
# you will get an error.
```

```
# IndexError: list index out of range
```

```
#print(my_tuple[6])
```

```
# index must be an integer
```

```
# If you uncomment line 21,
```

```
# you will get an error.
```

```
# TypeError: list indices must be integers, not float
```

```
#my_tuple[2.0]
```

```
# nested tuple
n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
```

```
# nested index
# Output: 's'
print(n_tuple[0][3])
```

```
# nested index
# Output: 4
print(n_tuple[1][1])
```

Slicing

Kita bisa mengakses berbagai item dalam tupel dengan menggunakan operator pengiris - titik dua ":".

```
my_tuple = ('p','r','o','g','r','a','m','i','z')
```

```
# elements 2nd to 4th
# Output: ('r', 'o', 'g')
print(my_tuple[1:4])
```

```
# elements beginning to 2nd
# Output: ('p', 'r')
print(my_tuple[:2])
```

```
# elements 8th to end
```

```
# Output: ('i', 'z')
```

```
print(my _tuple[7:])
```

```
# elements beginning to end
```

```
# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```

```
print(my _tuple[:])
```

Mengubah Tuple

Tidak seperti daftar, tupel tidak dapat diubah.

Ini berarti elemen tupel tidak dapat diubah begitu telah ditetapkan. Tapi, jika elemen itu sendiri adalah datatype yang bisa berubah seperti daftar, item nested-nya bisa diubah.

Kita juga bisa menugaskan tuple ke nilai yang berbeda (reassignment).

```
my _tuple = (4, 2, 3, [6, 5])
```

```
# we cannot change an element
```

```
# If you uncomment line 8
```

```
# you will get an error:
```

```
# TypeError: 'tuple' object does not support item assignment
```

```
#my _tuple[1] = 9
```

```
# but item of mutable element can be changed
```

```
# Output: (4, 2, 3, [9, 5])
```

```
my_tuple[3][0] = 9
```

```
print(my_tuple)
```

```
# tuples can be reassigned
```

```
# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```

```
my_tuple = ('p','r','o','g','r','a','m','i','z')
```

```
print(my_tuple)
```