Python Tuples Cheat Sheet

Learn computer science with Python:

www.k0nze.dev

Tuples

Python tuples are a sequential and immutable data type that allows organizing data in an ordered manner. Each element of a tuple can be of a different data type. However, because tuples are immutable, they can't be changed after declaring them.

Defining a Tuple

```
>>> t = (True, 42, 0.23, "Hi")
>>> t
  (True, 42, 0.23, 'Hi')
```

Retrieving Tuple Elements

```
>>> t = (True, 42, 0.23, "Hi")
>>> t[0]
  True
>>> t[1]
  42
>>> t[-1]
  "Hi"
>>> t[-2]
  0.23
```

Trying to Change a Tuple Element

```
>>> t = (True, 42, 0.23, "Hi")
>>> l[0] = False
  Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
  TypeError: 'tuple' object does not
  support item assignment
```

Tuple vs. Lists

Tuples are immutable in contrast to lists that are a mutable data type. Due to being immutable, tuples are generally faster than lists.

Lists	Tuples
Lists are mutable	Tuples are immutable
Iterating over all elements is time-consuming	Iterating over all elements is faster compared to lists
Lists consume more memory	Tuples consume less memory compared to lists
Lists have a lot of built-in functions	Tuples don't have a lot of built-in functions
Lists can be changed during the runtime of a program which can lead to errors	Tuples can not change which can help to avoid runtime errors

Tuple Constant Folding

Python's optimizer precomputes tuples of constants into a single constant. However, have to be built during runtime, which makes tuples of constants generally faster to access.

```
>>> from dis import dis
>>> dis(compile("(1,2,3)", '', 'eval'))
  1     0 LOAD_CONST      0 ((1,2,3))
        2 RETURN_VALUE
>>> dis(compile("[1,2,3]", '', 'eval'))
  1     0 BUILD_LIST      0
        2 LOAD_CONST      0 ((1,2,3))
        4 LIST_EXTEND     1
        6 RETURN_VALUE
```

Tuples Save Memory

Since tuples are immutable, their size is fixed and can not be changed. Therefore, Python only allocates the necessary amount of memory for tuple and does not over-allocate.

```
>>> import sys
>>> sys.getsizeof((1,2,3,4,5,6,7,8,9))
  112
>>> sys.getsizeof([1,2,3,4,5,6,7,8,9])
  152
```

Tuples are Faster to Index

Tuples reference to their elements directly, while lists and other Python objects have another layer indirection to access their elements.

```
$ python3 -m timeit -s 't = (1,2,3)' 't[2]'
  20000000 loops, best of 5:
  18.4 nsec per loop
$ python3 -m timeit -s 't = [1,2,3]' 't[2]'
  20000000 loops, best of 5:
  18.6 nsec per loop
$ python3 -m timeit -s 't=(1,2,3)' 'a,b,c=t'
  20000000 loops, best of 5:
  15.2 nsec per loop
$ python3 -m timeit -s 't=[1,2,3]' 'a,b,c=t'
  20000000 loops, best of 5:
  15.9 nsec per loop
```

Tuple Slicing

With the []-operator, it is possible to access single tuple elements and access sub tuples of a tuple in the same way as list slicing.

Syntax

```
t[start:stop:step]
```

Parameters

`start:` start index from which the sub tuple is returned
`stop:` stop index until the sub tuple is returned
`step:` which elements are included in the sub tuple (2 means every 2nd element is returned)

Retrieving Sub Lists

```
>>> t = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
>>> t[2:6]
  (2, 3, 4, 5)
>>> t[5:-1]
  (5, 6, 7, 8)
>>> t[1:7:2]
  (1, 3, 5)
>>> t[4:]
  (4, 5, 6, 7, 8, 9)
>>> t[:6]
  (0, 1, 2, 3, 4, 5)
>>> t[::3]
  (0, 3, 6, 9)
>>> t[::-1]
  (9, 8, 7, 6, 5, 4, 3, 2, 1, 0)
```

Tuple of Lists

Tuples can not only store primitive data types. Tuples can also store complex data types such as lists.

Creating a Tuple of Lists

```
>>> l = ['a','b']
>>> m = ['c','d']
>>> n = ['e','f']
>>> t = (l, m, n)
>>> t
(['a', 'b'], ['c', 'd'], ['e', 'f'])
```

Accessing List Elements in Tuples

```
>>> t[0][1]
'b'
>>> t[-1][-2]
'e'
>>> t[1][0] = 'z'
>>> t
(['a', 'b'], ['z', 'd'], ['e', 'f'])
>>> l.append('x')
>>> t
(['a', 'b', 'x'], ['z', 'd'], ['e', 'f'])
```

List of Tuples and Sorting

Tuples can be used as list elements.

Creating a List of Tuples

```
>>> s = (5, 8, 9)
>>> t = (3, 9, 6)
>>> u = (2, 6, 3)
>>> v = (5, 4, 9)
>>> w = (2, 5, 4)
>>> l = [s, t, u, v, w]
>>> l
  [(5,8,9),(3,9,6),(2,6,3),(5,4,9),(2,5,4)]
```

Sorting a List of Tuples

The default .sort() function of lists sorts a list of tuples using the > operator, which compares the tuples element-wise. That means the first element is the primary sorting key, the second element is the secondary sorting key, etc. This is also called lexicographical order:

```
>>> l.sort()
>>> l
  [(2,5,4),(2,6,3),(3,9,6),(5,4,9),(5,8,9)]
```

A list of tuples can also be sorted using other sorting keys such as the sum of all elements:

```
>>> l = [(4, -2), (-2, 6)]
>>> l.sort()
>>> l
[(-2, 6), (4, -2)]
>>> l.sort(key=sum)
>>> l
[(4, -2), (-2, 6)]
```

The key order for the sorting of tuple elements can also be changed using a lambda function that returns a tuple ordered in the sorting priority:

```
>>> >>> l =
  [(2,5,4),(2,6,3),(3,9,6),(5,4,9),(5,8,9)]
>>> l.sort(key=lambda t:(t[1],t[2],t[0]))
>>> l
  [(5,4,9),(2,5,4),(2,6,3),(5,8,9),(3,9,6)]
```

Tuples in `if,elif,else`

in Operator

The in operator can be used to check if an element is contained in a tuple.

```
>>> t = ('a','b','c','d')
>>> if 'z' in t:
...     print("found z")
... elif 'a' in t:
...     print("found a")
... else:
...     print("did not find a or z")
  found a
>>> x = 1 if 'a' in t else 0
>>> x
  1
```

Comparison Operators

Tuples can be compared to each other using the six comparision operators: `==,!=,<,>,>=,<=.`
The comparison operators compare tuples elementwise lexicographical order.

```
>>> t = (1, 2)
>>> s = (3, 4, 5)
>>> u = (6, 7)
>>> v = (1, 1, 1)
>>> w = (1, 2)
>>> t == s
  False
>>> t == u
  False
>>> t != s
   True
>>> t < s
   True
>>> t >= v
   True
>>> t == w
  True
```

Addition +

The plus (+) operator can add tuples together.

Syntax

```
tuple + tuple
```

Parameters

`tuple:` tuples that are added together

Return Value

None / result of tuple additon

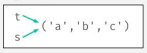
```
>>> t = ('a','b')
>>> s = ('c','d')
>>> t + s
  ('a','b','c','d')
```

Tuples can not be Copied

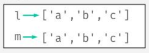
Tuples do not support the .copy() function. This is again because tuples are immutable. A copy of a Python object only makes sense when the copy should be changed, and the original should stay the same (or the other way around). Therefore only references for tuples exist.

```
>>> t = ('a', 'b', 'c')
>>> s = ('a', 'b', 'c'
>>> t is s
  True
```

Reference



Copy



`max() & min()`

Returns the maximum/minimum element of a tuple (iterable).

Syntax

```
max(iterable, key=None)
min(iterable, key=None)
```

Parameters

`iterable:` list (iterable) that is sorted
`key:` sorting key

Return Value

maximum/minimum element

```
>>> t = ('a', 'aa', 'b', 'c')
>>> max(t)
  'c'
>>> max(t, key=len)
  'aa'
>>> min(t)
  'a'
```

Tuples in Loops

The in operator can be used to loop over each element in a tuple:

for each loop

```
>>> t = ('a','b','c',d')
>>> for x in t:
...     print(x, end=',')
  a,b,c,d,
```

for loop over range

```
>>> t = ('a','b','c',d')
>>> for i in range(len(t)):
...     print(t[i], end=',')
  a,b,c,d,
```

for loop enumerate

```
>>> t = ('a','b','c',d')
>>> for i, x in enumerate(t):
...     print(f"t[{i}]={x}", end=',')
  t[0]=1,t[1]=2,t[2]=3,t[3]=4,
```

`.count()`

Returns how often a given value is contained in a tuple.

Syntax

```
tuple.count(value)
```

Parameters

`value:` value that is counted

Return Value

amount of how often value is in the tuple

```
>>> t = ['a','b','b','c']
>>> t.count('b')
  2
>>> t.count('z')
  0
```

`.index()`

Returns the index of the first occuring element with a given value from a tuple.

Syntax

```
tuple.index(value, start=0, stop=inf)
```

Parameters

`value:` value for which the index is returned
`start:` index from which the search for the given value starts
`stop:` index at which the search for the given value stops

Return Value

element that was removed from the tuple

```
>>> t = ('a','b','z','c','d',z','f')
>>> t.index('b')
  1
>>> t.index('z', 3)
  5
>>> t.index('z', 1, 3)
  2
```

`len()`

Returns the length of a list or any object that implements `__len__(self)`.

Syntax

```
len(object)
```

Parameters

`object:` object that implements `__len__(self)`

Return Value

lenght of the object

```
>>> t = ('a','b','c',d')
>>> len(t)
  4
```

Single Element Tuple

Tuples can also hold only a single element

Syntax

```
>>> t = ('a',)
>>> t
  ('a',)
```

k0nze builds
k0nze_
discord.k0nze.dev
patreon.com/k0nze

www.k0nze.dev