CS112
Introduction to Python Programming

Session 05: Tuples and Sets

Shengwei Hou
Ph.D., Assistant Professor
Department of Ocean Science and Engineering
Fall 2022

SUSTech Southern University of Science and Technology

明德求是 日新自强
VIRTUE | TRUTH | ADVANCE

Contents

SUSTech Southern University of Science and Technology

Tuples

- Basics: creation and operations
- Nested tuple
- Indexing and slicing
- Built-in functions
- Tuple and list/dict
- Tuple methods
- Packing/Unpacking
- The `zip()` function

Sets

- Basics
- Built-in functions
- Set operations
- Set methods
- Frozenset

01

Tuples

Tuples

- Tuples are lists that are immutable, i.e., once a tuple is created, you cannot change its values
- A tuple is a finite sequence of ordered, ~~immutable~~, and heterogeneous items that are of fixed size enclosed in `()`
- Tuples are very useful when returning more than one value from a function, or passing multiple values through one function parameter

```
>>> c = (1, 1, 2, 3, 5, 8, 13)
```

Tuples

- Individual items of a tuple are addressed in the same way as those of lists, but the elements cannot be changed:

```
>>> c = (1, 1, 2, 3, 5, 8, 13)

>>> c[4] = 7
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not
support item assignment
```

Tuple creation

- Tuples can be constructed ~~with or without~~ surrounding parentheses, although often parentheses are used:

```
>>> c = (1, 1, 2, 3)
>>> c
(1, 1, 2, 3)
>>> type(c)
<class 'tuple'>

>>> a = 1,2,3,4
>>> a
(1, 2, 3, 4)
>>> type(a)
<class 'tuple'>
```

Tuple creation

- Tuples can be created without any values:

```
>>> empty_tuple = ()
>>> empty_tuple
()
>>> type(empty_tuple)
<class 'tuple'>

>>> empty_tuple = tuple()
>>> empty_tuple
()
```

Tuple creation

- You can store any type of items, and even another tuple, in a tuple:

```
>>> air_force = ("f15", "f22a", "f35a")

>>> fighter_jets = (1988, 2005, 2016, air_force)

>>> fighter_jets
(1988, 2005, 2016, ('f15', 'f22a', 'f35a'))
```

Tuple creation

- Construction of tuples containing only one item:

```
>>> single_tuple1 = 'hello',

>>> single_tuple1
('hello',)

>>> single_tuple2 = ('hello',)

>>> single_tuple2
('hello',)
```

```
In [7]: s = ("test")
In [8]: s
Out[8]: 'test'
In [9]: type(s)
Out[9]: str

In [10]: s = ("test", )
In [11]: s
Out[11]: ('test',)
In [12]: type(s)
Out[12]: tuple
```

A tuple with one item should be constructed by having a value followed by a comma

Tuple operations

- Like in lists, you can use the + operator to concatenate tuples:

```
>>> tuple_1 = (2, 0, 1, 4)

>>> tuple_2 = (2, 0, 1, 9)

>>> tuple_1 + tuple_2
(2, 0, 1, 4, 2, 0, 1, 9)
```

a new tuple returned

Tuple operations

- You can also use the * operator to repeat a tuple:

```
>>> tuple_1 = (2, 0, 1, 4)

>>> tuple_2 = (2, 0, 1, 9)

>>> tuple_1 * 3
(2, 0, 1, 4, 2, 0, 1, 4, 2, 0, 1, 4)

>>> tuple_1 == tuple_2
False
```

Tuple operations

- You can check for the presence of an item in a tuple using `in` and `not in` membership operators:

```
>>> tuple_items = (1, 9, 8, 8)

>>> 1 in tuple_items
True

>>> 25 not in tuple_items
True
```

Tuple operations

- Comparison operators like <, <=, >, >=, == and != can also be used to compare tuples:

```
>>> tuple_1 = (9, 8, 7)

>>> tuple_2 = (9, 1, 1)

>>> tuple_1 > tuple_2
True

>>> tuple_1 != tuple_2
True
```

Tuples are compared position by position

The `tuple()` function

- The built-in `tuple()` function is used to create a tuple from an iterable object:

```
>>> tuple("sustech")
('s', 'u', 's', 't', 'e', 'c', 'h')
```

String to tuple

```
>>> tuple(range(10))
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Range to tuple

```
>>> tuple(['CS', '112'])
('CS', '112')
```

List to tuple

```
>>> tuple({'a':1,'b':2})
('a', 'b')
```

Dict to tuple

Nested tuple

- A tuple can nest other tuples:

```
>>> letters = ("a", "b", "c")
>>> numbers = (1, 2, 3)
>>> nested_tuples = (letters, numbers)
>>> nested_tuples
(('a', 'b', 'c'), (1, 2, 3))

>>> type(nested_tuples)
<class 'tuple'>
```

Indexing in tuples

- The index of tuples starts from 0, and should always be an integer:

```
>>> cities = ("Beijing", "Shanghai", "Guangzhou", "Shenzhen")

>>> cities[0]
'Beijing'

>>> cities[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: tuple index out of range

>>> cities[-1]
'Shenzhen'
```

Slicing in tuples


```
>>> colors = ("v", "i", "b", "g", "y", "o", "r")
>>> colors
('v', 'i', 'b', 'g', 'y', 'o', 'r')
>>> colors[1:4]
('i', 'b', 'g')
>>> colors[:5]
('v', 'i', 'b', 'g', 'y')
>>> colors[3:]
('g', 'y', 'o', 'r')
>>> colors[:]
('v', 'i', 'b', 'g', 'y', 'o', 'r')
```

Slicing in tuples


```
>>> colors = ("v", "i", "b", "g", "y", "o", "r")
>>> colors[::]
('v', 'i', 'b', 'g', 'y', 'o', 'r')
>>> colors[1:5:2]
('i', 'g')
>>> colors[::2]
('v', 'b', 'y', 'r')
>>> colors[::-1]
('r', 'o', 'y', 'g', 'b', 'i', 'v')
>>> colors[-5:-2]
('b', 'g', 'y')
```

Built-in functions for tuples

- `len()` function returns the numbers of items in a tuple
- `sum()` function returns the sum of numbers in the tuple
- `sorted()` function returns a sorted copy of the tuple as a list while leaving the original tuple untouched

```
>>> years = (1987, 1985, 1981, 1996)
>>> len(years)
4
>>> sum(years)
7949
>>> sorted_years = sorted(years)
>>> sorted_years
[1981, 1985, 1987, 1996]
```

In the case of string items in a tuple, they are sorted based on their ASCII values.

Relation between tuples and lists

- Lists are mutable, while tuples are immutable
- The `list()` function can convert a tuple to a list

```
>>> nucleotide = ("A", "C", "G", "T")
>>> nucleotide[3] = "U"
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item
assignment

>>> nucleotide_list = list(nucleotide)
>>> nucleotide_list
['A', 'C', 'G', 'T']
```

Relation between tuples and lists

- If an item within a tuple is mutable, then you can change it. Consider the presence of a list as an item in a tuple, then any changes to the list get reflected on the overall items in the tuple:

```
>>> german_cars = ["porsche", "audi", "bmw"]
>>> european_cars = ("ferrari", "volvo", "renault", german_cars)
>>> european_cars
('ferrari', 'volvo', 'renault', ['porsche', 'audi', 'bmw'])

>>> european_cars[3].append("mercedes")
>>> german_cars
['porsche', 'audi', 'bmw', 'mercedes']

>>> european_cars
('ferrari', 'volvo', 'renault', ['porsche', 'audi', 'bmw', 'mercedes'])
```

Relation between tuples and dicts

- Tuples can be used as `key:value` pairs to build dictionaries, which is achieved by nesting tuples within tuples, wherein each nested tuple item should have two items (the first item becomes the key and second item as its value):

```
>>> fish_weight = (("white_shark", 520), ("beluga", 1571),
("greenland_shark",1400))

>>> fish_weight_dict = dict(fish_weight)

>>> fish_weight_dict
{'white_shark': 520, 'beluga': 1571, 'greenland_shark': 1400}
```

Relation between tuples and dicts

- The method `items()` in a dictionary returns `dict_items()` that can be converted to ~~a list of tuples~~ where each tuple corresponds to a `key:value` pair of the dictionary:

```
>>> founding_year = {"Google":1996, "Apple":1976, "Sony":1946,
"ebay":1995, "IBM":1911}

>>> founding_year.items()
dict_items([('Google', 1996), ('Apple', 1976), ('Sony', 1946), ('ebay',
1995), ('IBM', 1911)])

>>> list(founding_year.items())
[('Google', 1996), ('Apple', 1976), ('Sony', 1946), ('ebay', 1995),
('IBM', 1911)]
```

Tuple methods

- The `count()` method counts the number of times the item has occurred in the tuple and returns it
- The `index()` method searches for the given item from the start of the tuple and returns its first appearance index

```
>>> channels = ("ngc", "discovery", "animal_planet",
"history", "ngc")

>>> channels.count("ngc")
2

>>> channels.index("history")
3
```

Tuple packing and unpacking

- The values `12345, 54321` and `'hello!'` are packed together into a tuple
- The reverse operation of tuple packing is also possible

```
>>> t = 12345, 54321, 'hello!'
>>> t
(12345, 54321, 'hello!')

>>> x, y, z = t
>>> x
12345
>>> y
54321
>>> z
'hello!'
```

Tuple packing and unpacking

- Tuple unpacking requires that there are as many variables on the left side of the equals sign as there are items in the tuple
- Multiple assignments are just a combination of tuple packing and unpacking, e.g.

```
>>> a, b=3, 4
>>> a
3
>>> b
4
```

```
In [1: t = 12345, 54321, 'hello!'
In [2]: t → 3 values
Out[2]: (12345, 54321, 'hello!')
In [3]: x,y,z=t
In [4]: x,y = t
---------------------
ValueError  Traceback (most recent call last)
<ipython-input-87-623752d04abb> in <module>
----> 1 x,y = t
      2 values ⇒ error.
ValueError: too many values to unpack (expected 2)
In [5]: x,y,_ = t
```

The `zip()` function

- Returns a sequence of tuples, where the i-th tuple contains the i-th element from each of the iterables. The aggregation of elements stops when the shortest input iterable is exhausted:

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6, 7]

>>> zipped = zip(x, y)
>>> zipped
<zip object at 0x0000022AA66C3BC8>

>>> list(zipped)
[(1, 4), (2, 5), (3, 6)]
```

02

Sets

Sets

- A set is an unordered collection with ~~no duplicate items~~
- Primary uses of sets include membership testing and eliminating duplicate entries
- Curly braces `{}` or the `set()` function can be used to create sets with a comma-separated list of items

```
>>> basket = {'apple', 'orange', 'apple', 'pear'}
>>> basket
{'orange', 'apple', 'pear'}
```

Sets

- To create an empty set, you must use `set()` but NOT `{}`, as the latter creates an empty dictionary

```
>>> type({})
<class 'dict'>

>>> type(set())
<class 'set'>
```

Sets

- Sets are mutable
- Indexing is not possible in sets, since set items are unordered
- You cannot access or change an item of the set using indexing or slicing

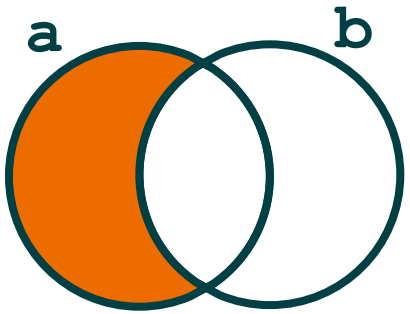
```
>>> basket = {'apple', 'orange', banana', 'pear'}
>>> basket[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable
>>> basket[0] = 'kiwi'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object does not support item assignment
```

Sets

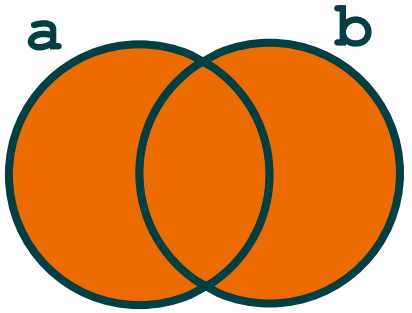

```
>>> basket = {'apple', 'orange', 'banana', 'pear'}

>>> 'orange' in basket                    in and not in
True
>>> 'crabgrass' not in basket
True

>>> len(basket)                           len()
4

>>> sorted(basket)                              sorted()
['apple', 'banana', 'orange', 'pear']
```

Set operations

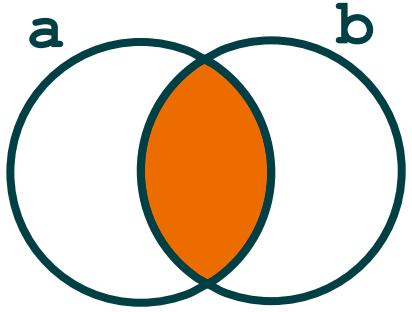

```
>>> a = set('abracadabra')
>>> a
{'a', 'c', 'd', 'b', 'r'}
>>> b = set('alacazam')
>>> b
{'a', 'm', 'c', 'z', 'l'}

>>> a - b
{'r', 'd', 'b'}
>>> a | b
{'a', 'm', 'c', 'd', 'b', 'z', 'l', 'r'}
>>> a & b
{'a', 'c'}
>>> a ^ b
{'m', 'd', 'z', 'l', 'r', 'b'}
```



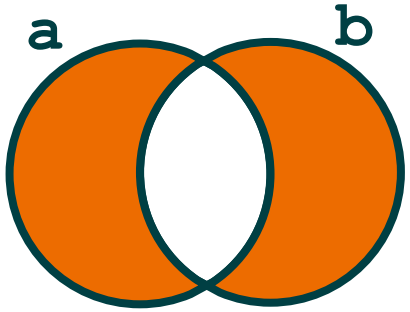
Difference
a-b



Union
a|b (A ∪ B)



Intersection
a&b (A ∩ B)



Symmetric difference
a^b (A Δ B)

Set methods


```
>>> dir(set)
['__and__', '__class__', '__contains__', '__delattr__',
'__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__gt__', '__hash__', '__iand__',
'__init__', '__init_subclass__', '__ior__', '__isub__',
'__iter__', '__ixor__', '__le__', '__len__', '__lt__',
'__ne__', '__new__', '__or__', '__rand__', '__reduce__',
'__reduce_ex__', '__repr__', '__ror__', '__rsub__',
'__rxor__', '__setattr__', '__sizeof__', '__str__',
'__sub__', '__subclasshook__', '__xor__', 'add', 'clear',
'copy', 'difference', 'difference_update', 'discard',
'intersection', 'intersection_update', 'isdisjoint',
'issubset', 'issuperset', 'pop', 'remove',
'symmetric_difference', 'symmetric_difference_update',
'union', 'update']
```

Set methods


```
>>> set1 = {"a", "b", "e", "f", "g"}
>>> set2 = {"a", "e", "c", "d"}

>>> set2.difference(set1)
{'c', 'd'}

>>> set2.intersection(set1)
{'a', 'e'}

>>> set2.union(set1)
{'a', 'c', 'd', 'b', 'e', 'f', 'g'}

>>> set2.symmetric_difference(set1)
{'c', 'g', 'd', 'b', 'f'}
```

Set operations (-, |, &, ^) can also be done with set methods

Set methods


```
>>> set1 = {"a", "b", "e", "f", "g"}
>>> set2 = {"a", "e", "c", "d"}

>>> set2.add("h")
>>> set2
{'h', 'a', 'c', 'd', 'e'}

>>> set2.update(set1)
>>> set2
{'h', 'b', 'f', 'a', 'c', 'd', 'g', 'e'}
```

Add an element to a set.

Update a set with the union of itself and others.

Set methods


```
>>> set1 = {"a", "b", "e", "f", "g"}

>>> set1.discard("a")
>>> set1
{'g', 'e', 'b', 'f'}

>>> set1.clear()
>>> set1
set()
```

Remove an element from a set if it is a member.
If the element is not a member, do nothing.

Remove all elements from this set.

Frozenset

- A frozenset is basically the same as a set, except that it is immutable. Once a frozenset is created, its items cannot be changed

```
>>> fs = frozenset(["g", "o", "o", "d"])
>>> fs
frozenset({'d', 'o', 'g'})

>>> animals = set([fs, "cattle", "horse"])
>>> animals
{'horse', frozenset({'d', 'o', 'g'}), 'cattle'}
```

Exercises

Exercise 1


```
fruits = ("apple", "banana", "cherry", "orange", "apple")
```

1. Use the correct syntax to print the first item in the **`fruits`** tuple
2. Use the correct syntax to print the number of items in the **`fruits`** tuple
3. Use negative indexing to print the last item in the tuple
4. Use a range of indexes to print the third, fourth, and fifth item in the tuple
5. Use the `.index()` method to get the index of "orange " in the **`fruits`** tuple
6. Use the `.count()` method to count the number of "apple" in the **`fruits`** tuple

Hint: Use **`help()`** or search the internet to find out how to use the `.index()` and `.count()` methods.

Exercise 2

Swap two numbers (obtained from keyboard input) without using intermediate/temporary variables

Example output 1:

```
$ python example.py
Enter the first number: 10
Enter the second number: 20
After swapping, the first number becomes 20, the second number becomes 10
```

Example output 2:

```
$ python example.py
Enter the first number: 66
Enter the second number: 99
After swapping, the first number becomes 99, the second number becomes 66
```

Exercise 3

Concatenate a tuple and a list.

```
>>> test_list = [5, 6, 7]
>>> test_tup = (9, 10)
```

Expected output: `(5, 6, 7, 9, 10)`

Exercise 4

Convert a dictionary to two tuples, one for all keys and one for all values.

```
founding_year = {"Google":1996, "Apple":1976,
"Sony":1946, "ebay":1995, "IBM":1911}
```

Expected output:

```
('Google', 'Apple', 'Sony', 'ebay', 'IBM')
(1996, 1976, 1946, 1995, 1911)
```

Exercise 5


```
fruits = {"apple", "banana", "cherry"}

more_fruits = ["orange", "mango", "grapes"]
```

1. Use the correct syntax to check if "apple" is present in the `fruits` set.
2. Use the correct method to add "orange" to the `fruits` set.
3. Use the correct method to add multiple items (`more_fruits`) to the `fruits` set.
4. Use the `.remove` method to remove "banana" from the `fruits` set.
5. Use the `.discard` method to remove "apple" from the `fruits` set.
6. Can you find out the difference between `.remove` and `.discard` ? (Hint: Use `help()` or search the internet)

Exercise 6

Find unique words in `sentence` and print the sorted unique words:

```
sentence = "The man we saw saw a saw"
```

Exercise 7

Check if a given string is binary string or not using set

```
Input: "01010101010"
Output: True

Input: "1"
Output: True

Input: "CS112"
Output: False
```

Exercise 8

Find out which vowels (`a, e, i, o, u`) are missing in the sentence. Note the sentence should be treated as case-insensitive here, which means `"a"` is the same as `"A"`.

```
Input: "Apple"
Output: {'u', 'i', 'o'}

Input: "Hello World"
Output: {'u', 'a', 'i'}

Input: "Southern University of Science and Technology"
Output: set()
```