Tuples
Sorting: optional parameter `key`
`lambda` functions

Tuples

non-scalar, compound types: composed of accessible small pieces

tuple: sequence of elements of any type (sequence type)
- **homogeneous tuple**: all items of the same type
- **heterogeneous tuple**: items of different type

Tuples are similar to lists but **tuples are immutable**

values: items in parenthesis (optional), separated by commas

```
>>> a1 = (9, 5, -78, 0, 56, -4)
>>> a2 = ('cat', 'dog', 'wolf')
>>> a3 = (9.5, 24, 'wolf', True, 'hi')

>>> emptytuple = ()
>>> t1 = (25,)
```

- homogeneous tuple of integers
- homogeneous tuple of strings
- heterogeneous tuple

- tuple with a single element:
  parenthesis: optional
  comma: compulsory

Basic operations

- + (concatenation), * (replication)
- membership operators: `in`, `not in`
- comparison operators: ==, !=

```
>>> a1 = ('cat', 4.5, 'wolf')
>>> a2 = ('caiman', 'crocodile', 83)
>>> a3 = ('cat', 4.5, 'wolf')
>>> a1 == a2
False
>>> a1 == a3
True
>>> a2 != a3
True
```

```
>>> a1 = (9, 8, 7)
>>> a2 = 2, 4, 6
>>> a1 + a2
(9, 8, 7, 2, 4, 6)
>>> a1*3
(9, 8, 7, 9, 8, 7, 9, 8, 7)
>>> 8 in a1
True
>>> 4 in a1
False
```

- parenthesis are opcional
- Python always uses parenthesis to represent a tuple

Indexing

indexing operator: `[]`

index: integer expression. Item **position**

Let `a` be a tuple:
`a[index]` item of `a` at position `index`

positive, **zero-based** indices: 0 <= index < len(a)
negative indices: -len(a) <= index < 0

```
>>> a1 = ('cat', 4.5, 'wolf', 2, 6)
>>> a1[0]
'cat'
>>> a1[3]
2
>>> a1[-1]
6
```

items:	'cat'	4.5	'wolf'	2	6
positive indices:	0	1	2	3	4
negative indices:	-5	-4	-3	-2	-1

Slicing

Let `a` be a tuple:
`a[start:stop:step]` slice (subtuple) of `a` from `start` (included) to `stop` (not included)
`step:` jumps step characters

positive indices: 0 <= start < stop < len(a)

Default values:
start: 0
stop: len(a)
step: 1

```
>>> a = ('caiman', 1.5, 'crocodile', 83, True, -29, False)
>>> a[1:4]
(1.5, 'crocodile', 83)
>>> a[:3]                          # default start = 0
('caiman', 1.5, 'crocodile')
>>> a[4:]                          # default stop = len(a)
(True, -29, False)
>>> a[::2]                 # default start = 0, stop = len(a)
('caiman', 'crocodile', True, False)
>>> a[::-1]                        # reversing
(False, -29, True, 83, 'crocodile', 1.5, 'caiman')
```

Nested tuples and lists

```
>>> a = ['caiman', (3, 9, 5), ('cat', [8, 9], 'cow', 'sheep')] # nested structure
>>> a[0]      # string
'caiman'
>>> a[1]      # homogeneous subtuple of integers
(3, 9, 5)
>>> a[1][1]
9
>>> a[2]      # heterogeneous subtuple
('cat', [8, 9], 'cow', 'sheep')
>>> a[2][3]   # string at position 3 in subtuple at position 2 of list a
'sheep'
# char at position 1 in string at position 3 in subtuple at position 2 of list a
>>> a[2][3][1]
'h'
# integer at position 0 in sublist at position 1 in subtuple at position 2 of list a
>>> a[2][1][0]
8
```

Tuples: built-in functions, methods

functions: `len`, `max`, `min`, `sum`, `sorted` tuple methods: `count`, `index`

```
>>> a = (99, 7, 76, -22, 0, 7)
>>> b = ('shark', 'dog', 'crocodile', 'sheep', 'cat')
>>> len(a)
6
>>> max(b)
'sheep'
>>> sum(a)
158
>>> sorted(a)
[-22, 0, 7, 7, 76, 99]
>>> tuple(sorted(b))
('cat', 'crocodile', 'dog', 'shark', 'sheep')
>>> a.count(7)
2
>>> b.index('cat')
4
```

- function `sorted` returns a list
- function `tuple` converts to a tuple

Tuples and multiple assignment

```
>>> m, n = 5, 6
>>> m, n
(5, 6)
>>> m
5
>>> n
6

>>> a = 3, 4
>>> a
(3, 4)
>>> x, y = a
>>> x, y
(3, 4)
>>> x
3
>>> y
4
```

- multiple assignment → tuple assignment

- tuple assignment: pack
- tuple assignment: unpack

Packing and unpacking apply to lists and strings as well

Tuples and functions with several return values

A Python function always returns a single value.

- function that returns nothing: returns the value `None`
- function that returns several values: returns a tuple

```
def example(a, b):
    return a+b, a*b

>>> sum, prod = example(4, 5)
>>> sum, prod
(9, 20)
>>> t1 = example(3, 8)
>>> t1
(11, 24)
```

- returns a 2 items tuple, parenthesis are optional
- unpacked tuple assignment
- tuple assignment

Optional parameters

Functions and methods may have optional parameters:

- optional: when calling the function or invoking the method we can avoid defining the actual parameter
- they have a default value
- for a value different from default, the parameter name must be specified

Function sorted has 2 optional parameters: `key` and `reverse`

```
>>> help(sorted)
Help on built-in function sorted in module builtins:

sorted(iterable, /, *, key=None, reverse=False)
  Return a new list containing all items from the iterable in ascending order.

  A custom key function can be supplied to customize the sort order, and the
  reverse flag can be set to request the result in descending order.
```

Optional parameter: `reverse`

Functions `sorted` and method `sort` have the optional parameter `reverse`

- default value `reverse` = `False`: sorting in ascending order
- `reverse` = `True`: sorting in descending order (parameter name specified)

function `sorted`

```
>>> ta = (7, 4, 8, 1)
>>> sorted(ta)
[1, 4, 7, 8]
>>> sorted(ta, reverse = True)
[8, 7, 4, 1]
```

function `sorted` takes a string, a list, a tuple or a dictionary but always returns a list

`list` method `sort`

```
>>> lb = ['wing', 'nose', 'leg', 'paw', 'ear']
>>> lb.sort()
>>> lb
['ear', 'leg', 'nose', 'paw', 'wing']
>>> lc = ['wing', 'nose', 'leg', 'paw', 'ear']
>>> lc.sort(reverse = True)
>>> lc
['wing', 'paw', 'nose', 'leg', 'ear']
```

Optional parameter: `key`

Optional parameter `key` specifies the sorting criterion.
It is a function of 1 parameter:

- An already defined function
- An anonymous function (`lambda` function)

Functions `max`, `min` and `sorted`
and method `sort` have parameter `key`

Already **defined function**.

Example:
function `len` (builtins library)
Sorting key: string length

```
>>> ta = ('wing', 'nose', 'paw', 'mouth')
>>> min(ta)
'mouth'
>>> min(ta, key= len)
'paw'
>>> max(ta)
'wing'
>>> max(ta, key= len)
'mouth'
>>> sorted(ta, key= len)
['paw', 'wing', 'nose', 'mouth']

>>> lb = ['wing', 'nose', 'paw', 'mouth']
>>> lb.sort(key = len)
>>> lb
['paw', 'wing', 'nose', 'mouth']
```

Optional parameter: `key`

Optional parameter key is a function of 1 parameter:
Functions `max`, `min` and sorted and method `sort` have parameter `key`

Already **defined function**.
Example: function `vowels_number`
defined in file `example.py`
Sorting key: number of vowels

```
def vowels_number(s):
  n = 0
  for c in s:
     if c in 'aeiouAEIOU':
       n = n + 1
   return n
```

```
>>> from example import vowels_number
>>> lb = ['Alice', 'Paul', 'Andrew', 'John', 'Martha', 'Pol']
>>> min(lb, key=vowels_number)
'John'
>>> max(lb, key=vowels_number)
'Alice'
>>> sorted(lb, key=vowels_number)
['John', 'Pol', 'Paul', 'Andrew', 'Martha', 'Alice']
```

Optional parameter: `key`

Optional parameter key is a function of 1 parameter:

- An **anonymous function (`lambda` function)**

Anonymous function syntax: `lambda` parameters: return expression

```
>>> def example1(x):
...     return 2*x+1
...
>>> anonym1 = lambda x: 2*x+1

>>> def example2(x, y):
...     return x + y
...
>>> anonym2 = lambda x, y: x+y
```

Optional parameter: `key`

Optional parameter key is a function of 1 parameter:

anonymous function: `lambda` function

Example: `lambda x: x.count('a') + x.count('A')`
Sorting key: number of 'a' letters, both lower and uppercase

`lambda` function

parameters `key` and `reverse`

```
>>> lb = ['Alice', 'Paul', 'Andrew', 'John', 'Martha', 'Pol']
>>> lb.sort(key=lambda x: x.count('a') + x.count('A'))
>>> lb
['John', 'Pol', 'Alice', 'Paul', 'Andrew', 'Martha']

>>> lb = ['Alice', 'Paul', 'Andrew', 'John', 'Martha', 'Pol']
>>> lb.sort(key=lambda x: x.count('a')+x.count('A'), reverse = True)
>>> lb
['martha', 'alice', 'paul', 'andrew', 'jonh', 'pol']
```

Sorting nested lists

Sorting by default (lexicographically):
Sort sublists, sub, first by name, sub[0] (`str`), then by age, sub[1] (`int`)

```
>>> lb = [['Ann', 22], ['Pol', 21], ['Emma', 21], ['Ella', 22], ['Mia', 20], ['Al', 23]]
>>> sorted(lb)
[['Al', 23], ['Ann', 22], ['Ella', 22], ['Emma', 21], ['Mia', 20], ['Pol', 21]]
```

Sort sublists, sub, first by age, sub[1] (`int`), then by name, sub[0] (`str`)

```
>>> lb = [['Ann', 22], ['Pol', 21], ['Emma', 21], ['Ella', 22], ['Mia', 20], ['Al', 23]]
>>> sorted(lb, key=lambda x: (x[1], x[0]))
[['Mia', 20], ['Emma', 21], ['Pol', 21], ['Ann', 22], ['Ella', 22], ['Al', 23]]
```

Creation of nested lists

Example: Write function identity that takes an integer, n, and returns a nested list representing an n-dimensional identity matrix. Examples:

```
>>> identity(2)
[[1, 0], [0, 1]]
>>> identity(4)
[[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]
```

Creation of nested lists

Example: Write function identity that takes an integer, n, and returns a nested list representing an n-dimensional identity matrix. Examples:

```
def identity(n):
  mat = []
  for i in range(n):

    row = []
    for j in range(n):
      if i == j:
        row.append(1)
      else:
        row.append(0)

    mat.append(row)

  return mat
```

Initialize the nested list

Create n rows

Initialize a row

Create n elements of a row

Creation process in the inner `for` similar to that in the outer `for`

Append each row to the nested list