ipaddress — Internet Addresses

Purpose: Classes for working with Internet Protocol (IP) addresses.

The `ipaddress` module includes classes for working with IPv4 and IPv6 network addresses. The classes support validation, finding addresses and hosts on a network, and other common operations.

Addresses

The most basic object represents the network address itself. Pass a string, integer, or byte sequence to `ip_address()` to construct an address. The return value will be a `IPv4Address` or `IPv6Address` instance, depending on the type of address being used.

```
# ipaddress_addresses.py

import binascii
import ipaddress


ADDRESSES = [
    '10.9.0.6',
    'fdfd:87b5:b475:5e3e:b1bc:e121:a8eb:14aa',
]

for ip in ADDRESSES:
    addr = ipaddress.ip_address(ip)
    print('{!r}'.format(addr))
    print('   IP version:', addr.version)
    print('   is private:', addr.is_private)
    print('  packed form:', binascii.hexlify(addr.packed))
    print('      integer:', int(addr))
    print()
```

Both classes can provide various representations of the address for different purposes, as well as answer basic assertions such as whether the address is reserved for multicast communication or if it is on a private network.

```
$ python3 ipaddress_addresses.py

IPv4Address('10.9.0.6')
   IP version: 4
   is private: True
  packed form: b'0a090006'
      integer: 168361990

IPv6Address('fdfd:87b5:b475:5e3e:b1bc:e121:a8eb:14aa')
   IP version: 6
   is private: True
  packed form: b'fdfd87b5b4755e3eb1bce121a8eb14aa'
      integer: 337611086560236126439725644408160982186
```

Networks

A network is defined by a range of addresses. It is usually expressed with a base address and a mask indicating which portions of the address represent the network, and which portions are remaining to represent addresses on that network. The mask can be expressed explicitly, or using a prefix length value as in the example below.

```
# ipaddress_networks.py

import ipaddress

NETWORKS = [
    '10.9.0.0/24',
```

```
    'fdfd:87b5:b475:5e3e::/64',
]

for n in NETWORKS:
    net = ipaddress.ip_network(n)
    print('{!r}'.format(net))
    print('     is private:', net.is_private)
    print('      broadcast:', net.broadcast_address)
    print('     compressed:', net.compressed)
    print('   with netmask:', net.with_netmask)
    print('  with hostmask:', net.with_hostmask)
    print('  num addresses:', net.num_addresses)
    print()
```

As with addresses, there are two network classes for IPv4 and IPv6 networks. Each class provides properties or methods for accessing values associated with the network such as the broadcast address and the addresses on the network available for hosts to use.

```
$ python3 ipaddress_networks.py

IPv4Network('10.9.0.0/24')
     is private: True
      broadcast: 10.9.0.255
     compressed: 10.9.0.0/24
   with netmask: 10.9.0.0/255.255.255.0
  with hostmask: 10.9.0.0/0.0.0.255
  num addresses: 256

IPv6Network('fdfd:87b5:b475:5e3e::/64')
     is private: True
      broadcast: fdfd:87b5:b475:5e3e:ffff:ffff:ffff:ffff
     compressed: fdfd:87b5:b475:5e3e::/64
   with netmask: fdfd:87b5:b475:5e3e::/ffff:ffff:ffff:ffff::
  with hostmask: fdfd:87b5:b475:5e3e::/::ffff:ffff:ffff:ffff
  num addresses: 18446744073709551616
```

A network instance is iterable, and yields the addresses on the network.

```
# ipaddress_network_iterate.py

import ipaddress

NETWORKS = [
    '10.9.0.0/24',
    'fdfd:87b5:b475:5e3e::/64',
]

for n in NETWORKS:
    net = ipaddress.ip_network(n)
    print('{!r}'.format(net))
    for i, ip in zip(range(3), net):
        print(ip)
    print()
```

This example only prints a few of the addresses, because an IPv6 network can contain far more addresses than fit in the output.

```
$ python3 ipaddress_network_iterate.py

IPv4Network('10.9.0.0/24')
10.9.0.0
10.9.0.1
10.9.0.2

IPv6Network('fdfd:87b5:b475:5e3e::/64')
fdfd:87b5:b475:5e3e::
fdfd:87b5:b475:5e3e::1
fdfd:87b5:b475:5e3e::2
```

Iterating over the network yields addresses, but not all of them are valid for hosts. For example, the base address of the network and the broadcast address are both included. To find the addresses that can be used by regular hosts on the network, use the `hosts()` method, which produces a generator.

```
# ipaddress_network_iterate_hosts.py

import ipaddress

NETWORKS = [
    '10.9.0.0/24',
    'fdfd:87b5:b475:5e3e::/64',
]

for n in NETWORKS:
    net = ipaddress.ip_network(n)
    print('{!r}'.format(net))
    for i, ip in zip(range(3), net.hosts()):
        print(ip)
    print()
```

Comparing the output of this example with the previous example shows that the host addresses do not include the first values produced when iterating over the entire network.

```
$ python3 ipaddress_network_iterate_hosts.py

IPv4Network('10.9.0.0/24')
10.9.0.1
10.9.0.2
10.9.0.3

IPv6Network('fdfd:87b5:b475:5e3e::/64')
fdfd:87b5:b475:5e3e::1
fdfd:87b5:b475:5e3e::2
fdfd:87b5:b475:5e3e::3
```

In addition to the iterator protocol, networks support the `in` operator to determine if an address is part of a network.

```
# ipaddress_network_membership.py

import ipaddress


NETWORKS = [
    ipaddress.ip_network('10.9.0.0/24'),
    ipaddress.ip_network('fdfd:87b5:b475:5e3e::/64'),
]

ADDRESSES = [
    ipaddress.ip_address('10.9.0.6'),
    ipaddress.ip_address('10.7.0.31'),
    ipaddress.ip_address(
        'fdfd:87b5:b475:5e3e:b1bc:e121:a8eb:14aa'
    ),
    ipaddress.ip_address('fe80::3840:c439:b25e:63b0'),
]


for ip in ADDRESSES:
    for net in NETWORKS:
        if ip in net:
            print('{}\nis on {}'.format(ip, net))
            break
    else:
        print('{}\nis not on a known network'.format(ip))
    print()
```

The implementation of `in` uses the network mask to test the address, so it is much more efficient than expanding the full list of addresses on the network.

```
$ python3 ipaddress_network_membership.py
```

```
$ python3 ipaddress_network_membership.py

10.9.0.6
is on 10.9.0.0/24

10.7.0.31
is not on a known network

fdfd:87b5:b475:5e3e:b1bc:e121:a8eb:14aa
is on fdfd:87b5:b475:5e3e::/64

fe80::3840:c439:b25e:63b0
is not on a known network
```

Interfaces

A network interface represents a specific address on a network and can be represented by a host address and a network prefix or netmask.

```
# ipaddress_interfaces.py

import ipaddress


ADDRESSES = [
    '10.9.0.6/24',
    'fdfd:87b5:b475:5e3e:b1bc:e121:a8eb:14aa/64',
]


for ip in ADDRESSES:
    iface = ipaddress.ip_interface(ip)
    print('{!r}'.format(iface))
    print('network:\n  ', iface.network)
    print('ip:\n  ', iface.ip)
    print('IP with prefixlen:\n  ', iface.with_prefixlen)
    print('netmask:\n  ', iface.with_netmask)
    print('hostmask:\n  ', iface.with_hostmask)
    print()
```

The interface object has properties to access the full network and address separately, as well as several different ways to express the interface and network mask.

```
$ python3 ipaddress_interfaces.py

IPv4Interface('10.9.0.6/24')
network:
   10.9.0.0/24
ip:
   10.9.0.6
IP with prefixlen:
   10.9.0.6/24
netmask:
   10.9.0.6/255.255.255.0
hostmask:
   10.9.0.6/0.0.0.255

IPv6Interface('fdfd:87b5:b475:5e3e:b1bc:e121:a8eb:14aa/64')
network:
   fdfd:87b5:b475:5e3e::/64
ip:
   fdfd:87b5:b475:5e3e:b1bc:e121:a8eb:14aa
IP with prefixlen:
   fdfd:87b5:b475:5e3e:b1bc:e121:a8eb:14aa/64
netmask:
   fdfd:87b5:b475:5e3e:b1bc:e121:a8eb:14aa/ffff:ffff:ffff:ffff::
hostmask:
   fdfd:87b5:b475:5e3e:b1bc:e121:a8eb:14aa/::ffff:ffff:ffff:ffff
```

See also

- Standard library documentation for ipaddress
- **PEP 3144** – IP Address Manipulation Library for the Python Standard Library
- An introduction to the ipaddress module
- Wikipedia: IP address – An introduction to IP addresses and networks.
- *Computer Networks (5th Edition)* – By Andrew S. Tanenbaum and David J. Wetherall. Published by Pearson, 2010. ISBN-10: 0132126958

Networking

socket — Network Communication

Quick Links

Addresses
Networks
Interfaces

This page was last updated 2016-12-27.

Navigation

Networking
socket — Network Communication

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example