Python for Coding Interviews

https://github.com/mmicu/python-for-coding-interviews

May 8, 2022

Introduction

This guide includes a list of several and useful `Python` data structures to know for coding interviews.

It is intended to show the main data structures incorporated in the language and their useful functions. More advance `Python` features will not be shown here.

Additional material:

Topic	Link
Time complexity	https://wiki.python.org/moin/TimeComplexity
Python collections	https://docs.python.org/3/library/collections.html

Topics

1. Primitive Types
2. Tuples
3. Lists
4. Strings
5. Stacks
6. Queues
7. Sets
8. Hash Tables
9. Heaps
10. Collections
    1. namedtuple
    2. defaultdict
    3. Counter
    4. OrderedDict

Primitive Types

1. Booleans (`bool`).
2. Integers (`int`).
3. Floats (`float`).
4. Strings (`str`).

```
# Define variables
>>> b, i, f, s = True, 12, 8.31, 'Hello, world!'
>>> type(b)  # <class 'bool'>
>>> type(i)  # <class 'int'>   ~ Unbounded
>>> type(f)  # <class 'float'> ~ Bounded
>>> type(s)  # <class 'str'>

# Type Conversion
>>> str(i)
'12'
>>> float(i)
12.0
>>> str(b)
'True'
>>> int('10')
10
>>> int('10a')  # `ValueError: invalid literal for int() with base 10: '10a'`

# Operations
>>> 2 * 2
4
>>> 2 * 2.
4.0
>>> 4 / 2
2.0
>>> 5 // 2  # `//` is the integer division
2
>>> 3 % 2
1

# `min` and `max`
>>> min(4, 2)
2
>>> max(21, 29)
29

# Some useful math functions
>>> abs(-1.2)
1.2
>>> divmod(9, 4)
(2, 1)
>>> 2 ** 3  # Equivalent to `pow(2, 3)`
8

# Math functions from the `math` package
>>> import math
>>> math.ceil(7.2)
8
>>> math.floor(7.2)
7
>>> math.sqrt(4)
2.0

# Pseudo lower and upper bounds
>>> float('-inf')  # Pseudo min-int
-inf
>>> float('inf')  # Pseudo max-int
inf
```

range and enumerate

```
# `range`
>>> list(range(3))  # Equivalent to `range(0, 3)`
[0, 1, 2]
>>> list(range(1, 10, 2))
[1, 3, 5, 7, 9]
>>> for i in range(3): print(i)
...
0
1
2
>>> for i in range(2, -1, -1): print(i)  # Equivalent to `reversed(range(3))`
...
2
1
0

# `enumerate`
>>> for i, v in enumerate(range(3)): print(i, v)
...
0 0
1 1
2 2
>>> for i, v in enumerate(range(3), start=10): print(i, v)
...
10 0
11 1
12 2
```

Tuples

```
>>> t = (1, 2, 'str')
>>> type(t)
<class 'tuple'>
>>> t
(1, 2, 'str')
>>> len(t)
3

>>> t[0] = 10  # Tuples are immutable: `TypeError: 'tuple' object does not support item assignment`

>>> a, b, c = t  # Unpacking
>>> a
1
>>> b
2
>>> c
'str'
>>> a, _, _ = t  # Unpacking: ignore second and third elements
>>> a
1
```

Lists

Python uses Timsort algorithm in sort and sorted (https://en.wikipedia.org/wiki/Timsort).

```
# Define a list
>>> l = [1, 2, 'a']
>>> type(l)  # <class 'list'>
>>> len(l)
3
>>> l[0]  # First element of the list
1
>>> l[-1]  # Last element of the list (equivalent to `l[len(l) - 1]`)
'a'

# Slicing
>>> l[:]  # `l[start:end]` which means `[start, end)`
[1, 2, 'a']
>>> l[0:len(l)]  # `start` is 0 and `end` is `len(l)` if omitted
[1, 2, 'a']

# Some useful methods
>>> l.append('b')  # `O(1)`
>>> l.pop()  # `O(1)` just for the last element
'b'
>>> l.pop(0)  # `O(n)` since list must be shifted
1
>>> l
[2, 'a']
>>> l.remove('a')  # `O(n)`
>>> l.remove('b')  # `ValueError: list.remove(x): x not in list`
>>> l
[2]
>>> l.index(2)  # It returns first occurrence (`O(n)`)
0
>>> l.index(12)  # `ValueError: 12 is not in list`

# More compact way to define a list
>>> l = [0] * 5
>>> l
[0, 0, 0, 0, 0]
>>> len(l)
5
>>> [k for k in range(5)]
[0, 1, 2, 3, 4]
>>> [k for k in reversed(range(5))]
[4, 3, 2, 1, 0]

# Compact way to define 2D arrays
>>> rows, cols = 2, 3
>>> m = [[0] * cols for _ in range(rows)]
>>> len(m) == rows
True
>>> all(len(m[k]) == cols for k in range(rows))
True
```

```
# Built-in methods
>>> l = [2, 1, 4, 3]
>>> len(l)
4
>>> min(l)
1
>>> max(l)
4
>>> sum(l)
10
>>> any(v == 4 for v in l)
True
>>> any(v == 5 for v in l)
False
>>> all(v > 0 for v in l)
True

# Sort list in-place (`sort`)
>>> l = [10, 2, 0, 1]
>>> l
[10, 2, 0, 1]
>>> l.sort()  # It changes the original list
>>> l
[0, 1, 2, 10]
>>> l.sort(reverse=True)  # It changes the original list
>>> l
[10, 2, 1, 0]

# Sort a list a return a new one (`sorted`)
>>> l = [10, 2, 0, 1]
>>> sorted(l)  # It returns a new list
[0, 1, 2, 10]
>>> l  # Original list is not sorted
[10, 2, 0, 1]

# Sort by a different key
>>> students = [
...     ('Mark', 21),
...     ('Luke', 20),
...     ('Anna', 18),
... ]
>>> sorted(students, key=lambda s: s[1])  # It returns a new list
[('Anna', 18), ('Luke', 20), ('Mark', 21)]
>>> students.sort(key=lambda s: s[1])  # In-place
>>> students
[('Anna', 18), ('Luke', 20), ('Mark', 21)]
```

Strings

```
>>> s = 'Hello, world!'
>>> type(s)  # <class 'str'>
>>> len(s)
13
```

```
>>> s[0] = 'h'  # Strings are immutable: `TypeError: 'str' object does not support item assignment`
>>> s += ' Another string'  # A new string will be created, so concatenation is quite slow

>>> s = 'Hello'
>>> l = list(s)
>>> l
['H', 'e', 'l', 'l', 'o']
>>> l[0] = 'h'
>>> ''.join(l)
'hello'

>>> 'lo' in s
True
>>> ord('a')
97
>>> chr(97)
'a'
```

Stacks

```
>>> stack = []  # We can use a normal list to simulate a stack

>>> stack.append(0)  # `O(1)`
>>> stack.append(1)
>>> stack.append(2)

>>> len(stack)
3

>>> stack[0]  # Bottom of the stack
0
>>> stack[-1]  # Top of the stack
2

>>> stack.pop()  # `O(1)`
2
>>> stack.pop()
1

>>> len(stack)
1

>>> stack.pop()
0

>>> stack.pop()  # `IndexError: pop from empty list`
>>> stack[-1]    # `IndexError: pop from empty list`
```

Queues

```
>>> from collections import deque

>>> queue = deque()
```

```
# Enqueue -> append()
>>> queue.append(0)  # `O(1)`
>>> queue.append(1)
>>> queue.append(2)

>>> len(queue)
3

>>> queue[0]  # Head of the queue
0
>>> queue[-1]  # Tail of the queue
2

# Dequeue -> popleft()
>>> queue.popleft()  # `O(1)`
0
>>> queue.popleft()
1

>>> len(queue)
2

>>> queue.popleft()
2

>>> len(queue)
0

>>> queue.popleft()  # `IndexError: pop from an empty deque`
>>> queue[0]  # `IndexError: pop from an empty deque`
```

Sets

```
>>> s = set()
>>> s.add(1)
>>> s.add(2)
>>> s
{1, 2}
>>> len(s)
2
>>> s.add(1)  # Duplicate elements are not allowed per definition
>>> s
{1, 2}
>>> s.add('a')  # We can mix types
>>> s
{1, 2, 'a'}
>>> 1 in s  # `O(1)`
True
>>> s.remove(1)
>>> s
{2, 'a'}
>>> s.remove(1)  # `KeyError: 1`
>>> s.pop()  # Remove and return an arbitrary element from the set
2
```

```
>>> s0 = {1, 2, 'a'}
>>> s0
{1, 2, 'a'}
>>> s1 = set([1, 2, 'a'])
>>> s1
{1, 2, 'a'}

>>> s0 = {1, 2}
>>> s1 = {1, 3}
>>> s0 | s1
{1, 2, 3}
>>> s0.union(s1)  # New set will be returned
{1, 2, 3}

>>> s0 = {1, 2}
>>> s1 = {1, 3}
>>> s0 & s1
{1}
>>> s0.intersection(s1)  # New set will be returned
{1}

>>> s0 = {1, 2}
>>> s1 = {1, 3}
>>> s0 - s1
{2}
>>> s0.difference(s1)
{2}
```

Hash Tables

```
>>> d = {'a': 'hello, world', 'b': 11}  # Equivalent to `dict(a='hello, world', b=11)`
>>> type(d)
<class 'dict'>
>>> d
{'a': 'hello, world', 'b': 11}

>>> d.keys()
dict_keys(['a', 'b'])
>>> d.values()
dict_values(['hello, world', 11])
>>> for k, v in d.items():
...     print(k, v)
...
a hello, world
b 11

>>> 'a' in d  # `O(1)`
True
>>> 1 in d
False
>>> d['a'] += '!'
>>> d
{'a': 'hello, world!', 'b': 11}
```

```
>>> d[1] = 'a new element'
>>> d
{'a': 'hello, world!', 'b': 11, 1: 'a new element'}

>>> d[0] += 10  # `KeyError: 0`
>>> d.get(0, 1)  # Return `1` as default value since key `0` does not exist
1
>>> d.get(1, '?')  # Key `1` exists, so the actual value will be returned
'a new element'
>>> d.get(10) is None
True
```

Heaps

The following commands show how to work with a `min heap`. Currently, `Python` does not have public methods for the `max heap`. You can overcome this problem by applying one of the following strategies:

1. Invert the value of each number. So, for example, if you want to add 1, 2 and 3 in the min heap, you can `heappush` -3, -2 and -1. When you `heappop` you invert the number again to get the proper value. This solution clearly works if your domain is composed by numbers >= 0.
2. Invert your object comparison.

```
>>> import heapq

>>> min_heap = [3, 2, 1]
>>> heapq.heapify(min_heap)
>>> min_heap
[1, 2, 3]

>>> min_heap = []
>>> heapq.heappush(min_heap, 3)  # `O(log n)`
>>> heapq.heappush(min_heap, 2)
>>> heapq.heappush(min_heap, 1)

>>> min_heap
[1, 3, 2]
>>> len(min_heap)
>>> min_heap[0]
1
>>> heapq.heappop(min_heap)  # `O(log n)`
1
>>> min_heap
[2, 3]

>>> heapq.heappop(min_heap)
2
>>> heapq.heappop(min_heap)
3
>>> heapq.heappop(min_heap)  # `IndexError: index out of range`
```

collections

Container datatypes (collections package).

collections.namedtuple

```
>>> from collections import namedtuple

>>> Point = namedtuple('Point', 'x y')

>>> p0 = Point(1, 2)
>>> p0
Point(x=1, y=2)
>>> p0.x
1
>>> p0.y
2

>>> p1 = Point(x=1, y=2)
>>> p0 == p1
True

# Python >= 3.6.1
>>> from typing import NamedTuple
>>>
>>> class Point(NamedTuple):
...     x: int
...     y: int
...

>>> p0 = Point(1, 2)
>>> p1 = Point(x=1, y=2)
>>> p0 == p1
True
```

collections.defaultdict

```
>>> from collections import defaultdict

>>> d = defaultdict(int)
>>> d['x'] += 1
>>> d
defaultdict(<class 'int'>, {'x': 1})
>>> d['x'] += 2
>>> d
defaultdict(<class 'int'>, {'x': 3})
>>> d['y'] += 10
>>> d
defaultdict(<class 'int'>, {'x': 3, 'y': 10})

>>> d = defaultdict(list)
>>> d['x'].append(1)
>>> d['x'].append(2)
>>> d
defaultdict(<class 'list'>, {'x': [1, 2]})
```

collections.Counter

```
>>> from collections import Counter

>>> c = Counter('abcabcaa')
>>> c
Counter({'a': 4, 'b': 2, 'c': 2})
>>> c.keys()
dict_keys(['a', 'b', 'c'])
>>> c.items()
dict_items([('a', 4), ('b', 2), ('c', 2)])
>>> for k, v in c.items():
...     print(k, v)
...
a 4
b 2
c 2
>>> c['d']  # It acts as a `defaultdict` for missing keys
0
```

collections.OrderedDict

```
>>> from collections import OrderedDict

>>> d = OrderedDict()

>>> d['first'] = 1
>>> d['second'] = 2
>>> d['third'] = 3
>>> d
OrderedDict([('first', 1), ('second', 2), ('third', 3)])

>>> for k, v in d.items():
...     print(k, v)
...
first 1
second 2
third 3
```