pydata Documentation

0.1.0

gfjiang

2018 08 21

Contents

CHAPTER 1

python pandasnumpypython

python python python

1.1 3.1

pythonpython

1.1.1

python

```
>>> tup = 4, 5, 6
>>> tup
(4, 5, 6)
```

```
>>> nested_tup = (4, 5, 6), (7, 8)

>>> nested_tup
((4, 5, 6), (7, 8))
```

1.1.2

[]list

```
>>> a_list = [2, 3, 7, None]
>>> tup = ('foo', 'bar', 'baz')
>>> b_list = list(tup)
>>> b_list
['foo', 'bar', 'baz']
>>> b_list[1] = 'peekaboo'
```

(continues on next page)

()

```
    >>> b_list
    ['foo', 'peekaboo', 'baz']
```

(semantically)(interchangeably)

```
In [42]: gen = range(10)
In [43]: gen
Out[43]: range(0, 10)
In [44]: list(gen)
Out[44]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

append

insert

0(inclusive)

collection.deque

+

listextendlist:

listsort(in-plance)list

```
In [61]: a = [7, 2, 5, 1, 3]

In [62]: a.sort()

In [63]: a
Out[63]: [1, 2, 3, 5, 7]
```

sort(come in handy). –

```
In [64]: b = ['saw', 'small', 'He', 'foxes', 'six']

In [65]: b.sort(key=len)

In [66]: b
Out[66]: ['He', 'saw', 'six', 'small', 'foxes']
```

sorted

(Binary search)

bisect bisect.bisectbisect.insort

```
In [67]: import bisect

In [68]: c = [1, 2, 2, 2, 3, 4, 7]

In [69]: bisect.bisect(c, 2)
Out[69]: 4

In [70]: bisect.bisect(c, 5)
Out[70]: 6

In [71]: bisect.insort(c, 6)

In [72]: c
Out[72]: [1, 2, 2, 2, 3, 4, 6, 7]
```

bisectlist list

(notation)start:stop[]

```
In [73]: seq = [7, 2, 3, 7, 5, 6, 0, 1]
In [74]: seq[1:5]
Out[74]: [2, 3, 7, 5]
```

```
In [75]: seq[3:4] = [6, 3]
In [76]: seq
Out[76]: [7, 2, 3, 6, 3, 5, 6, 0, 1]
```

startstopstop-start

startstop(omit)

```
In [77]: seq[:5]
Out[77]: [7, 2, 3, 6, 3]

In [78]: seq[3:]
Out[78]: [6, 3, 5, 6, 0, 1]
```

(negative indices)

```
In [79]: seq[-4:]
Out[79]: [5, 6, 0, 1]

In [80]: seq[-6:-2]
Out[80]: [6, 3, 5, 6]
```

RMATLAB 3-1

:

```
In [81]: seq[::2]
Out[81]: [7, 3, 3, 6, 1]
```

-1listtuple:

```
In [82]: seq[::-1]
Out[82]: [1, 0, 6, 5, 3, 6, 3, 2, 7]
```

1.1.3

Python(a handful of)

enumerate

:

```
i = 0
for value in collection:
        # do something with value
        i += 1
```

pythonenumerate(i, value):

```
for i, value in enumerate(collection):
        # do something with value
```

enumeratedict:

```
In [83]: some_list = ['foo', 'bar', 'baz']

In [84]: mapping = {}
In [85]: for i, v in enumerate(some_list):
        ....: mapping[v] = i

In [86]: mapping
Out[86]: {'bar': 1, 'baz': 2, 'foo': 0}
```

sorted

sortedlist:

```
In [87]: sorted([7, 1, 2, 6, 0, 3, 2])
Out[87]: [0, 1, 2, 2, 3, 6, 7]

In [88]: sorted('horse race')
Out[88]: [' ', 'a', 'c', 'e', 'e', 'h', 'o', 'r', 'r', 's']
```

sortedsort

zip

zip:

```
In [89]: seq1 = ['foo', 'bar', 'baz']
In [90]: seq2 = ['one', 'two', 'three']

In [91]: zipped = zip(seq1, seq2)

In [92]: list(zipped)
Out[92]: [('foo', 'one'), ('bar', 'two'), ('baz', 'three')]
```

zip:

```
In [93]: seq3 = [False, True]

In [94]: list(zip(seq1, seq2, seq3))
Out[94]: [('foo', 'one', False), ('bar', 'two', True)]
```

zipenumerate(simultaneously):

```
In [95]: for i, (a, b) in enumerate(zip(seq1, seq2)):
         ....: print('{0}: {1}, {2}'.format(i, a, b))
         ....:
0: foo, one
1: bar, two
2: baz, three
```

"zipped"zip"unzip" listlist:

```
In [96]: pitchers = [('Nolan', 'Ryan'), ('Roger', 'Clemens'),
         ....: ('Schilling', 'Curt')]

In [97]: first_names, last_names = zip(*pitchers)

In [98]: first_names
Out[98]: ('Nolan', 'Roger', 'Schilling')

In [99]: last_names
Out[99]: ('Ryan', 'Clemens', 'Curt')
```

reversed

reversed:

```
In [100]: list(reversed(range(10)))
Out[100]: [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

reversedlistfor

1.1.4

dictpython -python(curly braces):

```
In [101]: empty_dict = {}

In [102]: d1 = {'a' : 'some value', 'b' : [1, 2, 3, 4]}
```

(continues on next page)

()

```
In [103]: d1
Out[103]: {'a': 'some value', 'b': [1, 2, 3, 4]}
```

:

```
In [104]: d1[7] = 'an integer'

In [105]: d1
Out[105]: {'a': 'some value', 'b': [1, 2, 3, 4], 7: 'an integer'}

In [106]: d1['b']
Out[106]: [1, 2, 3, 4]
```

:

```
In [107]: 'b' in d1
Out[107]: True
```

delpop:

```
In [108]: d1[5] = 'some value'

In [109]: d1
Out[109]:
{'a': 'some value',
'b': [1, 2, 3, 4],
7: 'an integer',
5: 'some value'}

In [110]: d1['dummy'] = 'another value'

In [111]: d1
Out[111]:
{'a': 'some value',
'b': [1, 2, 3, 4],
7: 'an integer',
5: 'some value',
'dummy': 'another value'}

In [112]: del d1[5]

In [113]: d1
Out[113]:
{'a': 'some value',
'b': [1, 2, 3, 4],
7: 'an integer',
'dummy': 'another value'}

In [114]: ret = d1.pop('dummy')

In [115]: ret
Out[115]: 'another value'

In [116]: d1
Out[116]: {'a': 'some value', 'b': [1, 2, 3, 4], 7: 'an integer'}
```

keysvalues :

```
In [117]: list(d1.keys())
Out[117]: ['a', 'b', 7]

In [118]: list(d1.values())
Out[118]: ['some value', [1, 2, 3, 4], 'an integer']
```

update:

```
In [119]: d1.update({'b' : 'foo', 'c' : 12})

In [120]: d1
Out[120]: {'a': 'some value', 'b': 'foo', 7: 'an integer', 'c': 12}
```

update

:

```
mapping = {}
for key, value in zip(key_list, value_list):
mapping[key] = value
```

(since)dict(essentially)2dict2:

```
In [121]: mapping = dict(zip(range(5), reversed(range(5))))

In [122]: mapping
Out[122]: {0: 4, 1: 3, 2: 2, 3: 1, 4: 0}
```

dict comprehensions

:

```
if key in some_dict:
        value = some_dict[key]
else:
        value = default_value
```

getpopif-else:

```
value = some_dict.get(key, default_value)
```

keygetNonepop dict :

```
In [123]: words = ['apple', 'bat', 'bar', 'atom', 'book']

In [124]: by_letter = {}
In [125]: for word in words:
.....:          letter = word[0]
.....:          if letter not in by_letter:
.....:                  by_letter[letter] = [word]
.....:          else:
```

(continues on next page)

()

```
.....:                     by_letter[letter].append(word)
.....:
In [126]: by_letter
Out[126]: {'a': ['apple', 'atom'], 'b': ['bat', 'bar', 'book']}
```

setdefault(**preceding**)for:

```
for word in words:
        letter = word[0]
        by_letter.setdefault(letter, []).append(word)
```

collectionsdefaultdict :

```
from collections import defaultdict
by_letter = defaultdict(list)
for word in words:
        by_letter[word[0]].append(word)
```

(**scalar**)(int, float, string) (hashability) hash:

```
In [127]: hash('string')
Out[127]: 5023931463650008331

In [128]: hash((1, 2, (2, 3)))
Out[128]: 1079636502276347782

In [129]: hash((1, 2, [2, 3])) # fails because lists are mutable
---------------------------------------------------------------------------
TypeError Traceback (most recent call last)
<ipython-input-129-800cd14ba8be> in <module>()
----> 1 hash((1, 2, [2, 3])) # fails because lists are mutable
TypeError: unhashable type: 'list'
```

listtuple:

```
In [130]: d = {}

In [131]: d[tuple([1, 2, 3])] = 5

In [132]: d
Out[132]: {(1, 2, 3): 5}
```

1.1.5

set set{}:

```
In [133]: set([2, 2, 2, 1, 3, 3])
Out[133]: {1, 2, 3}

In [134]: {2, 2, 2, 1, 3, 3}
Out[134]: {1, 2, 3}
```

set(intersection,)(symmetric difference):

```
In [135]: a = {1, 2, 3, 4, 5}

In [136]: b = {3, 4, 5, 6, 7, 8}
```

unionl:

```
In [137]: a.union(b)
Out[137]: {1, 2, 3, 4, 5, 6, 7, 8}

In [138]: a | b
Out[138]: {1, 2, 3, 4, 5, 6, 7, 8}
```

&intersection:

```
In [139]: a.intersection(b)
Out[139]: {3, 4, 5}

In [140]: a & b
Out[140]: {3, 4, 5}
```

3-1.

Function	Alternative syntax	Description
a.add(x)	N/A	Add element x to the set a
a.clear()	N/A	Reset the set a to an empty state, discarding all of its elements
a.remove(x)	N/A	Remove element x from the set a
a.pop()	N/A	Remove an arbitrary element from the set a, raising KeyError if the set is empty
a.union(b)	a | b	All of the unique elements in a and b
a.update(b)	a |= b	Set the contents of a to be the union of the elements in a and b
a.intersection(b)	a & b	All of the elements in *both* a and b
a.intersection_update(b)	a &= b	Set the contents of a to be the intersection of the elements in a and b
a.difference(b)	a - b	The elements in a that are not in b
a.difference_update(b)	a -= b	Set a to the elements in a that are not in b
a.symmetric_difference(b)	a ^ b	All of the elements in either a or b but *not both*
a.symmetric_difference_update(b)	a ^= b	Set a to contain the elements in either a or b but *not both*
a.issubset(b)	N/A	True if the elements of a are all contained in b
a.issuperset(b)	N/A	True if the elements of b are all contained in a
a.isdisjoint(b)	N/A	True if a and b have no elements in common

(in-place counterparts) :

```
In [141]: c = a.copy()

In [142]: c |= b

In [143]: c
Out[143]: {1, 2, 3, 4, 5, 6, 7, 8}

In [144]: d = a.copy()

In [145]: d &= b

In [146]: d
Out[146]: {3, 4, 5}
```

:

```
In [147]: my_data = [1, 2, 3, 4]

In [148]: my_set = {tuple(my_data)}

In [149]: my_set
Out[149]: {(1, 2, 3, 4)}
```

:

```
In [150]: a_set = {1, 2, 3, 4, 5}

In [151]: {1, 2, 3}.issubset(a_set)
Out[151]: True

In [152]: a_set.issuperset({1, 2, 3})
Out[152]: True
```

:

```
In [153]: {1, 2, 3} == {3, 2, 1}
Out[153]: True
```

1.1.6 (comprehensions)

python :

```
[expr for val in collection if condition]
```

for:

```
result = []
for val in collection:
        if condition:
                result.append(expr)
```

2:

```
In [154]: strings = ['a', 'as', 'bat', 'car', 'dove', 'python']
```

(continues on next page)

()

```
In [155]: [x.upper() for x in strings if len(x) > 2]
Out[155]: ['BAT', 'CAR', 'DOVE', 'PYTHON']
```

(**idiomatically**) :

```
dict_comp = {key-expr : value-expr for value in collection if condition}
```

:

```
set_comp = {expr for value in collection if condition}
```

(mostly)(similarly) :

```
In [156]: unique_lengths = {len(x) for x in strings}

In [157]: unique_lengths
Out[157]: {1, 2, 3, 4, 6}
```

map:

```
In [158]: set(map(len, strings))
Out[158]: {1, 2, 3, 4, 6}
```

:

```
In [159]: loc_mapping = {val : index for index, val in enumerate(strings)}

In [160]: loc_mapping
Out[160]: {'a': 0, 'as': 1, 'bat': 2, 'car': 3, 'dove': 4, 'python': 5}
```

:

```
In [161]: all_data = [['John', 'Emily', 'Michael', 'Mary', 'Steven'],
.....:  ['Maria', 'Juan', 'Javier', 'Natalia', 'Pilar']]
```

'e' :

```
names_of_interest = []
for names in all_data:
        enough_es = [name for name in names if name.count('e') >= 2]
        names_of_interest.extend(enough_es)
```

(wrap):

```
In [162]: result = [name for names in all_data for name in names if name.count('e') >
↪= 2]

In [163]: result
Out[163]: ['Steven']
```

(a bit hard to wrap your head around) for(arrange) "(flantten)":

```
In [164]: some_tuples = [(1, 2, 3), (4, 5, 6), (7, 8, 9)]

In [165]: flattened = [x for tup in some_tuples for x in tup]

In [166]: flattened
Out[166]: [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

forfor:

```
flattened = []

for tup in some_tuples:
        for x in tup:
                flattened.append(x)
```

23 (**?**):

```
In [167]: [[x for x in tup] for tup in some_tuples]
Out[167]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

1.2 3.2

1.2.1

1.2.2

1.2.3

1.2.4 Lambda

1.2.5

1.2.6

1.2.7

1.3 3.3

1.3.1 Unicode

1.4 3.4

CHAPTER 2

numpy

python pandasnumpypython

python python python

2.1 4.1 NumPy ndarray

2.2 4.2 (element-wise)

2.3 4.3

2.4 4.4

2.5 4.5

2.6 4.6 (pseudorandom)

2.7 4.7

2.8 4.8