**PYTHON NOTES**

>>> bin(0)

'0b0'

>>> bin(-1)

'-0b1'

>>> ~0

-1

* Keys in dictionary can be of only hashable type.

**------------------------------------------------------------------------------------------------**

>>> a = range(3)

>>> \*b, = a

>>> c = \*a,

>>> b

[0, 1, 2]

>>> c

(0, 1, 2)

>>> \*d, = zip(b,c)

>>> d

[(0, 0), (1, 1), (2, 2)]

Above works for Python 3.5+.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

## itertools module

**>>>** **from** itertools **import** **\***

**>>>** a **=** combinations**(**'abcd'**,**2**)**

**>>>** list**(**a**)**

**[(**'a'**,** 'b'**),** **(**'a'**,** 'c'**),** **(**'a'**,** 'd'**),** **(**'b'**,** 'c'**),** **(**'b'**,** 'd'**),** **(**'c'**,** 'd'**)]**

**>>>** a **=** combinations\_with\_replacement**(**'abcd'**,**2**)**

**>>>** list**(**a**)**

**[(**'a'**,** 'a'**),** **(**'a'**,** 'b'**),** **(**'a'**,** 'c'**),** **(**'a'**,** 'd'**),** **(**'b'**,** 'b'**),** **(**'b'**,** 'c'**),** **(**'b'**,** 'd'**),** **(**'c'**,** 'c'**),** **(**'c'**,** 'd'**),** **(**'d'**,** 'd'**)]**

**>>>** a **=** permutations**(**'abcd'**,**2**)**

**>>>** list**(**a**)**

**[(**'a'**,** 'b'**),** **(**'a'**,** 'c'**),** **(**'a'**,** 'd'**),** **(**'b'**,** 'a'**),** **(**'b'**,** 'c'**),** **(**'b'**,** 'd'**),** **(**'c'**,** 'a'**),** **(**'c'**,** 'b'**),** **(**'c'**,** 'd'**),** **(**'d'**,** 'a'**),** **(**'d'**,** 'b'**),** **(**'d'**,** 'c'**)]**

**>>>** list**(**product**(**'abcd'**))**

**[(**'a'**,),** **(**'b'**,),** **(**'c'**,),** **(**'d'**,)]**

**>>>** list**(**product**(**'abcd'**,** repeat **=** 2**))**

**[(**'a'**,** 'a'**),** **(**'a'**,** 'b'**),** **(**'a'**,** 'c'**),** **(**'a'**,** 'd'**),** **(**'b'**,** 'a'**),** **(**'b'**,** 'b'**),** **(**'b'**,** 'c'**),** **(**'b'**,** 'd'**),** **(**'c'**,** 'a'**),** **(**'c'**,** 'b'**),** **(**'c'**,** 'c'**),** **(**'c'**,** 'd'**),** **(**'d'**,** 'a'**),** **(**'d'**,** 'b'**),** **(**'d'**,** 'c'**),** **(**'d'**,** 'd'**)]**

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

**>>>** accumulate**(**iterable**[,**func**])**

Make an iterator that returns accumulated sums. Elements may be any addable type. If the optional func argument is supplied, it should be a function of two arguments and it will be used instead of addition.

**>>>** a **=** accumulate**(**range**(**5**))**

**>>>** list**(**a**)**

**[**0**,** 1**,** 3**,** 6**,** 10**]**

**>>>** **import** operator

**>>>** a **=** accumulate**(**range**(**5**),**operator**.**mul**)**

**>>>** list**(**a**)**

**[**0**,** 0**,** 0**,** 0**,** 0**]**

**>>>** a **=** accumulate**(**range**(**1**,**5**),**operator**.**mul**)**

**>>>** list**(**a**)**

**[**1**,** 2**,** 6**,** 24**]**

**>>>** data **=** **[**3**,**4**,**6**,**2**,**1**,**9**,**0**,**7**,**5**,**8**]**

**>>>** list**(**accumulate**(**data**,** max**))**

**[**3**,** 4**,** 6**,** 6**,** 6**,** 9**,** 9**,** 9**,** 9**,** 9**]**

**>>>** data **=** 'abcd'

**>>>** list**(**accumulate**(**data**))**

**[**'a'**,** 'ab'**,** 'abc'**,** 'abcd'**]**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**>>>** a **=** chain**(**range**(**5**),**'abcd'**)**

**>>>** list**(**a**)**

**[**0**,** 1**,** 2**,** 3**,** 4**,** 'a'**,** 'b'**,** 'c'**,** 'd'**]**

**>>>** a **=** chain**(**range**(**5**),** 'abcd'**,[**'sam'**,**'ram'**])**

**>>>** list**(**a**)**

**[**0**,** 1**,** 2**,** 3**,** 4**,** 'a'**,** 'b'**,** 'c'**,** 'd'**,** 'sam'**,** 'ram'**]**

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

**>>>** list**(**chain**.**from\_iterable**([[**1**],[**2**]]))**

**[**1**,** 2**]**

**>>>** list**(**chain**.**from\_iterable**(**'abcd'**))**

**[**'a'**,** 'b'**,** 'c'**,** 'd'**]**

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

**>>> compress(data,selector) →** Make an iterator that filters elements from data returning only those that have a corresponding element in selectors that evaluates to True. Stops when either the data or selectors iterables has been exhausted.

**>>>** list**(**compress**(**range**(**5**),[**11001**]))**

**[**0**]**

**>>>** list**(**compress**([**0**,**1**,**2**,**3**,**4**],[**1**,**1**,**0**,**0**,**1**]))**

**[**0**,** 1**,** 4**]**

**>>>** list**(**compress**(**range**(**5**),[**1**,**1**,**0**,**0**,**1**]))**

**[**0**,** 1**,** 4**]**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**>>>** a **=** dropwhile**(lambda** x**:** x **<** 5**,** **[**1**,**2**,**4**,**5**,**4**,**3**])**

**>>>** list**(**a**)**

**[**5**,** 4**,** 3**]**

Make an iterator that drops elements from the iterable as long as the predicate is true; afterwards, returns every element. Note, the iterator does not produce any output until the predicate first becomes false.

**>>>** a **=** takewhile**(lambda** x**:** x **<**5**,** **[**1**,**2**,**3**,**5**,**6**,**3**,**2**])**

**>>>** list**(**a**)**

**[**1**,** 2**,** 3**]**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

**>>>** a **=** filterfalse**(lambda** x**:** x**<**5**,** **[**1**,**2**,**4**,**5**,**4**,**3**])**

**>>>** list**(**a**)**

**[**5**]**

Make an iterator that filters elements from iterable returning only those for which the predicate is False. If predicate is None, return the items that are false.

**>>>** a **=** filterfalse**(None,** **[**1**,**2**,**3**,**4**,False,True])**

**>>>** list**(**a**)**

**[False]**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

Starmap(func, iter)

>>> a = starmap(pow,[(2,3),(3,2)])

>>> list(a)

[8, 9]

>>> def func(x,y):

return str(x)+str(y)

>>> a = starmap(func,[(1,2),('a','b')])

>>> list(a)

['12', 'ab']

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

**Islice(iter, stop)**

**Islice(iter, start,stop[,step])**

**>>>** a **=** islice**(**'abcdefg'**,**2**)**

**>>>** list**(**a**)**

**[**'a'**,** 'b'**]**

**>>>** list**(**islice**(**'abcdefg'**,**1**,**3**))**

**[**'b'**,** 'c'**]**

**>>>** list**(**islice**(**'abcdefg'**,**1**,**5**,**2**))**

**[**'b'**,** 'd'**]**

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

**>>>** a **=** cycle**(**'ab'**)**

**>>>** **for** i **in** range**(**5**):**

next**(**a**)**

'a'

'b'

'a'

'b'

'a'

**>>>** a **=** count**(**10**,**2**)**

**>>>** **for** i **in** range**(**5**):**

next**(**a**)**

10

12

14

16

18

**>>>** a **=** repeat**(**'a'**,**3**)**

**>>>** list**(**a**)**

**[**'a'**,** 'a'**,** 'a'**]**

**~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~**

## random module

>>> import random

>>> [d for d in dir(random) if not '\_' in d]

['BPF', 'LOG4', 'Random', 'SystemRandom', 'TWOPI', 'betavariate', 'choice', 'choices', 'expovariate', 'gammavariate', 'gauss', 'getrandbits', 'getstate', 'lognormvariate', 'normalvariate', 'paretovariate', 'randint', 'random', 'randrange', 'sample', 'seed', 'setstate', 'shuffle', 'triangular', 'uniform', 'vonmisesvariate', 'weibullvariate']

>>> random.random() # return random no between 0.0(inclusive) and 1.0(exclusive)

0.22775387601856434

>>> random.random()

0.5177041206449146

>>> random.randint(2,4) #return random int between 2,4 (both inclusive)

3

>>> random.randint(2,4)

2

>>> random.randint(2,4)

2

>>> random.randrange(1,5)

4

Signature: random.randrange(start, stop=None, step=1, \_int=<class 'int'>)

Choose a random item from range(start, stop[, step]). This fixes the problem with randint() which includes the

endpoint; in Python this is usually not what you want.

>>> t = [1,2,3,5,67,3]

>>> random.choice(t) #return a random element from given collection

1

>>> random.choice(t)

3

>>> random.choice(t)

2

>>> random.uniform(2,4) #return a float from given range (both inclusive)

2.0900073372486214

>>> random.uniform(2,4)

2.5086430002165043

>>> random.sample(t,2) #return 2 elements from a collection

[5, 3]

## SQLAlchemy ORM

**from** sqlalchemy **import** create\_engine**,** Column**,** Integer**,** String**,** ForeignKey**,** Table**,** Text

**from** sqlalchemy**.**ext**.**declarative **import** declarative\_base

**from** sqlalchemy**.**orm **import** sessionmaker**,**relationship

engine **=** create\_engine**(**'sqlite:///blog.db'**,** echo **=** **True)**

Base **=** declarative\_base**()**

**class** **User(**Base**):**

\_\_tablename\_\_ **=** 'users'

id **=** Column**(**Integer**,** primary\_key **=** **True)**

name **=** Column**(**String**)**

fullname **=** Column**(**String**)**

password **=** Column**(**String**)**

**def** \_\_repr\_\_**(**self**):**

**return** "<User(name = '%s', fullname = '%s', password = '%s')>" **%(**self**.**name**,** self**.**fullname**,** self**.**password**)**

**class** **Address(**Base**):**

\_\_tablename\_\_ **=** 'addresses'

id **=** Column**(**Integer**,** primary\_key **=** **True)**

email\_address **=** Column**(**String**,** nullable **=** **False)**

user\_id **=** Column**(**Integer**,** ForeignKey**(**'users.id'**))**

user **=** relationship**(**'User'**,** back\_populates **=** 'addresses'**)**

# Address.user will return User instances associated with a email-address

**def** \_\_repr\_\_**(**self**):**

**return** "<Address(email\_address='%s')>" **%** self**.**email\_address

User**.**addresses **=** relationship**(**"Address"**,** order\_by**=**Address**.**id**,** back\_populates**=**"user"**)**

#-> User.addresses will return email-addresses associated with a given User instance.

post\_keywords **=** Table**(**'post\_keywords'**,** Base**.**metadata**,**\

Column**(**'post\_id'**,**ForeignKey**(**'posts.id'**),** primary\_key **=** **True),**\

Column**(**'keyword\_id'**,** ForeignKey**(**'keywords.id'**),** primary\_key **=** **True))**

**class** **BlogPost(**Base**):**

\_\_tablename\_\_ **=** 'posts'

id **=** Column**(**Integer**,** primary\_key **=** **True)**

user\_id **=** Column**(**Integer**,** ForeignKey**(**'users.id'**))**

headline **=** Column**(**String**(**255**),** nullable **=** **False)**

body **=** Column**(**Text**)**

keywords **=** relationship**(**'Keyword'**,** secondary**=**post\_keywords**,** back\_populates**=**'posts'**)**

**def** \_\_init\_\_**(**self**,** headline**,** body**,** author**):**

self**.**headline **=** headline

self**.**body **=** body

self**.**author **=** author

**def** \_\_repr\_\_**(**self**):**

**return** "<BlogPost(%r , %r, %r)>" **%** **(**self**.**headline**,** self**.**body**,** self**.**author**)**

**class** **Keyword(**Base**):**

\_\_tablename\_\_ **=** 'keywords'

id **=** Column**(**Integer**,** primary\_key **=** **True)**

keyword **=** Column**(**String**(**55**),** nullable **=** **False,** unique **=** **True)**

posts **=** relationship**(**'BlogPost'**,** secondary**=**post\_keywords**,** back\_populates**=**'keywords'**)**

**def** \_\_init\_\_**(**self**,** keyword**):**

self**.**keyword **=** keyword

**def** \_\_repr\_\_**(**self**):**

**return** "<Keyword(%r)>" **%** self**.**keyword

BlogPost**.**author **=** relationship**(**User**,** back\_populates **=** 'posts'**)**

User**.**posts **=** relationship**(**BlogPost**,** back\_populates **=**'author'**)**

Base**.**metadata**.**create\_all**(**engine**)**

Session **=** sessionmaker**(**bind **=** engine**)**

session **=** Session**()**

'''

session.add\_all([

User(name = 'sam', fullname = 'sam james', password = 'x'),

User(name = 'pete', fullname = 'pete james', password = 'y'),

User(name = 'max', fullname = 'max mac', password = 'y')])

session.new

session.commit()

a = session.query(User)

for i in a:

print i

<User(name = 'sam', fullname = 'sam james', password = 'x')>

<User(name = 'pete', fullname = 'pete james', password = 'y')>

<User(name = 'max', fullname = 'max mac', password = 'y')>

a = session.query(User).filter\_by(name = 'sam')

list(a)

<User(name = 'sam', fullname = 'sam james', password = 'x')>

a = session.query(User).filter\_by(password = 'y')

list(a)

[<User(name = 'pete', fullname = 'pete james', password = 'y')>,

<User(name = 'max', fullname = 'max mac', password = 'y')>]

for u in session.query(User).order\_by(User.id)[2:3]:

print u

<User(name = 'pete', fullname = 'pete james', password = 'y')>

for fullname in session.query(User.fullname):

print fullname

(u'sam james',)

(u'pete james',)

(u'max mac',)

Relationship ->

a = session.query(User)

b = list(a)[1] -> assigns '2nd' user to 'b'.

b.addresses -> [] #empty list as there are currently no email addresses for this user.

b.addresses = [Address(email\_address = 'x@gmail.com'), Address(email\_address = 'y@gmail.com')]

b.addresses -> returns list of email addresses

m = session.query(Address)

n1 = list(m)[0] -> assigned first email address to n1

n1.user -> returns associated user

n2 = list(m)[1] -> assigned second email address to n2

n2.user -> returns associated user

n1.user\_id -> returns 2

n2.user\_id -> returns 2

b.id -> returns 2

b.id is n1.user\_id -> returns True

b.id is n2.user\_id -> returns True

here ForeignKey 'user\_id' is same as id in User class.

That is why 'b.id is n1.user\_id' returns True

'''

**from** sqlalchemy **import** create\_engine**,** Column**,** Integer**,** String**,** ForeignKey**,** Table**,** Text

**from** sqlalchemy**.**ext**.**declarative **import** declarative\_base

**from** sqlalchemy**.**orm **import** sessionmaker**,** relationship

engine **=** create\_engine**(**'sqlite:///blog.db'**,** echo **=** **False)**

Base **=** declarative\_base**()**

**class** **User1(**Base**):**

\_\_tablename\_\_ **=** 'users1'

id **=** Column**(**Integer**,** primary\_key **=** **True)**

name **=** Column**(**String**)**

fullname **=** Column**(**String**)**

password **=** Column**(**String**)**

posts **=** relationship**(**'BlogPost1'**,** back\_populates **=**'author'**)**

addresses **=** relationship**(**"Address1"**,** back\_populates**=**"user"**)**

**def** \_\_repr\_\_**(**self**):**

**return** "<User(name = '%s', fullname = '%s', password = '%s')>" **%(**self**.**name**,** self**.**fullname**,** self**.**password**)**

**class** **Address1(**Base**):**

\_\_tablename\_\_ **=** 'addresses1'

id **=** Column**(**Integer**,** primary\_key **=** **True)**

email\_address **=** Column**(**String**,** nullable **=** **False)**

user\_id **=** Column**(**Integer**,** ForeignKey**(**'users1.id'**))**

user **=** relationship**(**'User1'**,** back\_populates **=** 'addresses'**)**

**def** \_\_repr\_\_**(**self**):**

**return** "<Address(email\_address='%s')>" **%** self**.**email\_address

post\_keywords1 **=** Table**(**'post\_keywords1'**,** Base**.**metadata**,**\

Column**(**'post\_id'**,**ForeignKey**(**'posts1.id'**),** primary\_key **=** **True),**\

Column**(**'keyword\_id'**,** ForeignKey**(**'keywords1.id'**),** primary\_key **=** **True))**

**class** **BlogPost1(**Base**):**

\_\_tablename\_\_ **=** 'posts1'

id **=** Column**(**Integer**,** primary\_key **=** **True)**

user\_id **=** Column**(**Integer**,** ForeignKey**(**'users1.id'**))**

headline **=** Column**(**String**(**255**),** nullable **=** **False)**

body **=** Column**(**Text**)**

author **=** relationship**(**'User1'**,** back\_populates **=** 'posts'**)**

keywords **=** relationship**(**'Keyword1'**,** secondary**=**post\_keywords1**,** back\_populates**=**'posts'**)**

**def** \_\_repr\_\_**(**self**):**

**return** "<BlogPost(%r , %r, %r)>" **%** **(**self**.**headline**,** self**.**body**,** self**.**author**)**

**class** **Keyword1(**Base**):**

\_\_tablename\_\_ **=** 'keywords1'

id **=** Column**(**Integer**,** primary\_key **=** **True)**

keyword **=** Column**(**String**(**55**),** nullable **=** **False,** unique **=** **True)**

posts **=** relationship**(**'BlogPost1'**,** secondary**=**post\_keywords1**,** back\_populates**=**'keywords'**)**

**def** \_\_repr\_\_**(**self**):**

**return** "<Keyword(%r)>" **%** self**.**keyword

Base**.**metadata**.**create\_all**(**engine**)**

Session **=** sessionmaker**(**bind **=** engine**)**

session **=** Session**()**

'''

This program is exactly similiar to program at sqltut1.py(above).

Both these programs share same database file(blog.db) but database of one file

is independent of other.

Unlike in sqltut1.py, we didn't use \_\_init\_\_ function because tutorial indicated that

in case of declarative mapping, using \_\_init\_\_ is optional.

Also, unlike sqltut1.py, statement like User.posts = relationship(.....) have been

incorporated in class definition as against to outside of class.'''

------------------------------------------