[Playhouse, extensions to Peewee — peewee 3.14.4 documentation (peewee-orm.com)](http://docs.peewee-orm.com/en/latest/peewee/playhouse.html" \l "dataset)

You can insert, update or delete using the dictionary APIs as well:

huey = table.find\_one(name='Huey')

*# {'age': 3, 'gender': None, 'id': 1, 'name': 'Huey'}*

*# Perform an update by supplying a partial record of changes.*

table[1] = {'gender': 'male', 'age': 4}

print(table[1])

*# {'age': 4, 'gender': 'male', 'id': 1, 'name': 'Huey'}*

*# Or insert a new record:*

table[3] = {'name': 'Zaizee', 'age': 2}

print(table[3])

*# {'age': 2, 'gender': None, 'id': 3, 'name': 'Zaizee'}*

*# Or delete a record:*

**del** table[3] *# Remove the row we just added.*

*# Import data from a CSV file into a new table. Columns will be automatically*

*# created for each field in the CSV file.*

new\_table = db['stats']

new\_table.thaw(format='csv', filename='monthly\_stats.csv')

Functional Programming:

[itertools — Functions creating iterators for efficient looping — Python 3.9.4 documentation](https://docs.python.org/3/library/itertools.html#itertool-functions)

Example of using [itemgetter()](https://docs.python.org/3/library/operator.html" \l "operator.itemgetter" \o "operator.itemgetter) to retrieve specific fields from a tuple record:

>>>

**>>>** inventory = [('apple', 3), ('banana', 2), ('pear', 5), ('orange', 1)]

**>>>** getcount = itemgetter(1)

**>>>** list(map(getcount, inventory))

[3, 2, 5, 1]

**>>>** sorted(inventory, key=getcount)

[('orange', 1), ('banana', 2), ('apple', 3), ('pear', 5)]

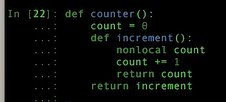
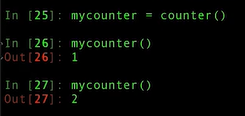
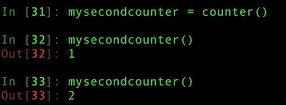
Currying

In [functional programming languages](https://en.wikipedia.org/wiki/Functional_programming_language), and many others, it provides a way of automatically managing how arguments are passed to functions and exceptions.

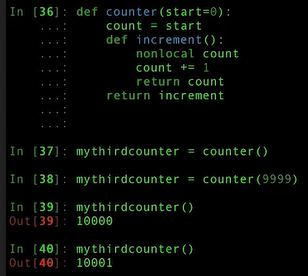
Composition functions:

z = f(g(x));

Closure is nested function.

Can create two counters from one closure.

So counter can operate independently. Like classes and objects

*The venerable master Qc Na was walking with his student, Anton. Hoping to prompt the master into a discussion, Anton said “Master, I have heard that objects (and classes) are a very good thing. Is this true?” Qc Na looked pityingly at his student and replied, “Foolish pupil! Objects are merely a poor man’s closures.”*

*Chastised, Anton took his leave from his master and returned to his cell, intent on studying closures. He carefully read the entire “Lambda: The Ultimate…” series of papers and its cousins, and implemented a small Scheme interpreter with a closure-based object system. He learned much, and looked forward to informing his master of his progress.*

*On his next walk with Qc Na, Anton attempted to impress his master by saying “Master, I have diligently studied the matter, and now understand that objects are truly a poor man’s closures.” Qc Na responded by hitting Anton with his stick, saying “When will you learn? Closures are a poor man’s object.” At that moment, Anton became enlightened.*

* Objects have methods.
* Closures are methods — they are defined and behave like functions, but like object methods they carry and internal state and take it into account when returning results.
* Objects can, and generally do, carry a mutable state.
* Closures can, and often do, carry a mutable state.
* Objects control access to their attributes — their internal state — through Properties and Python’s lexical scoping rules. By default, however, object attributes are externally accessible.
* Closures by nature tend to close around their internal state and thereby prevent external access, thus in terms of access to internal state, internal attributes, this is the opposite of the default behavior of an object. In accordance with Python’s Consenting Adults policy, a closure’s internal state is still accessible via its \_\_closure\_\_() dunder, but this violates the spirit of a closure — so do so at your own risk.

Thus, objects (or classes) and closures are similar, but they are not the same.

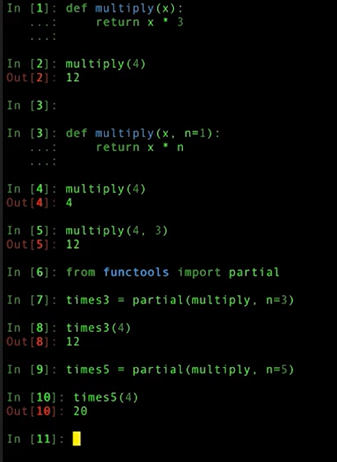
1 def closure(internal\_state):

2 def return\_function(arguments):

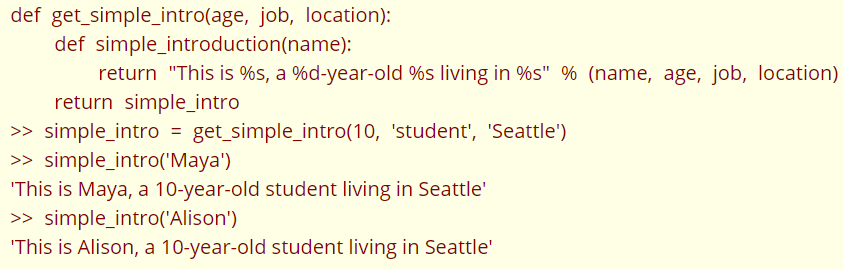
3 return internal\_state combined with arguments

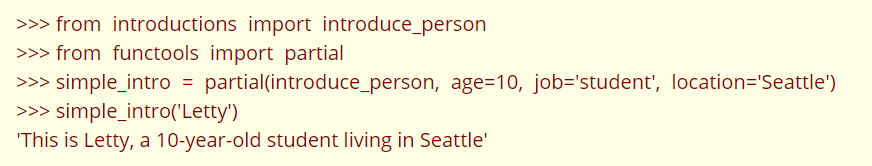
4 return return\_function

Currying objective is to reduce number of arguments (arity) in function

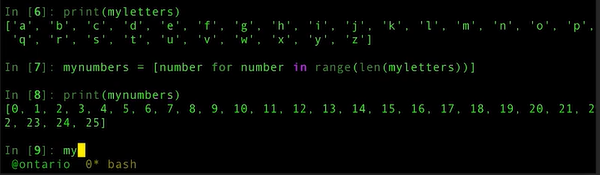


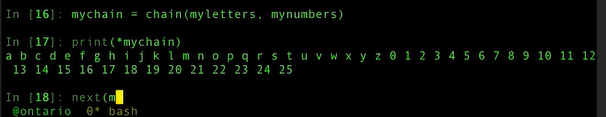
Currying can be used for prime key???

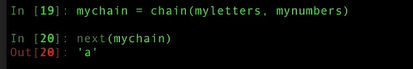


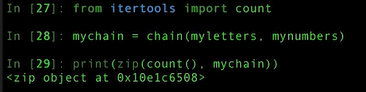


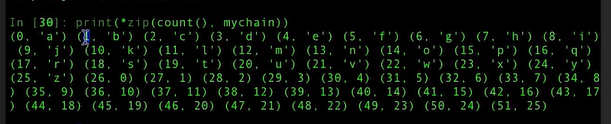
### Itertools

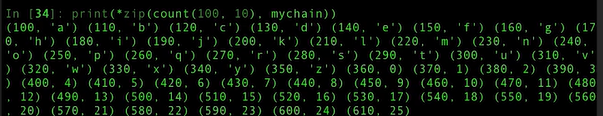


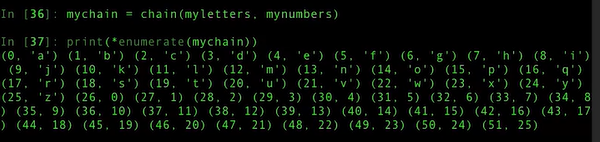












Tee – create multiply iterators at the same time

