Dir() Function: The dir() function is used to display the defined symbols.

Monkey patch in python: In python, the term monkey patch only refers to dynamic modifications of a class or module at run-time.

check this example : #Todo~

def test\_monthly\_schedule(self):

with patch('employee.request.get') as monkey:

monkey.return\_value.ok = True

monkey.return\_value.text = 'success'

xrange generate generator(iterator) and range generate list(iterable)

pickling and Unpickling::

Pickling: is the process whereby a Python object heirarchy is converted into a byte stream, and

Unpickling : is the inverse operation, whereby a byte stream is converted back into an object heirarchy.

pickling and unpickling alternatively known as serialization, marshalling and fattening

example:

import pickle

data1 = {'a': [1, 2.0, 3, 4+6j],

'b': ('string', u'Unicode string'),

'c': None}

selfref\_list = [1, 2, 3]

selfref\_list.append(selfref\_list)

#pickle dictionary using protocol 0

output = open('data1.pkl', 'wb')

pickle.dump(data1, output)

#Pickle the list using highest protocol available

pickle.dump(selfref\_list, output, -1)

output.close()

To read from a pickled file::

import pprint, pickle

pkl\_file = open('data1.pkl', 'rb')

data1 = pickle.load(pkl\_file)

pprint.pprint(data1)

data2 = pickle.load(pkl\_file)

pprint.pprint(data2)

pkl\_file.close()

this is often done if you want to save state between run or if you want to cache

the result of long running operations.

another use of pickling is if you need to transmit this dictionary or python objects over a network (using sockets etc). You first need to convert it into a character stream, then you can send it over a socket connection.

pickling is absolutely necessary for distributed and parallel computing.

=======================================

In python, by default all the parameters are passed "by reference" to the functions.

==================

Accessors and mutators are often called getters and setters in lang like "java".

For example: If x is a property of a use-defined class, then the class would have methods called set(x) and get(x) . Python has an @property decorator that allows you to add getter and setters in order to access the attribute of the class.

==========================================

In python, module is the way to structure program. Each python program file is a module, which imports other modules like objects and attributes

the folder of python program is a pcakage of modules. A package can have modules or sub folders

==========================

Magic Methods

\_\_call\_\_:

class Funky:

def \_\_call\_\_(self):

print("look at me, i work like a function")

>> f = Funky()

>> f()

Look at me, i work like a function

\_\_slots\_\_::

A class with \_\_slots\_\_ will create instances that dont have a \_\_dict\_\_

(they use the little less memory). A side effect of this is that instances cannot have other fields that what was specified in \_\_slots\_\_

\_\_new\_\_::

the \_\_new\_\_ method is the constructor(it returns the new instance) while \_\_init\_\_ is just a initializer (the instance is already created when instance is called)

class Foobar:

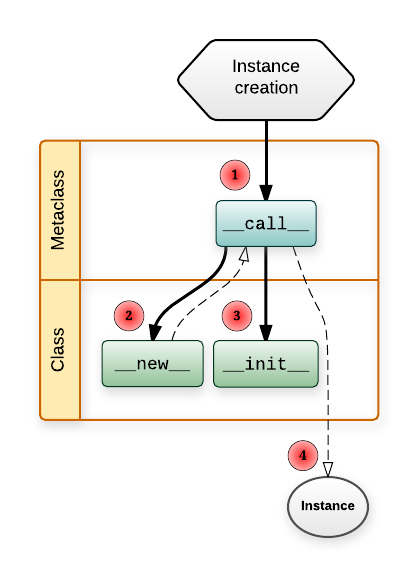
def \_\_new\_\_(self):

return super().\_\_new\_\_(cls)

Foobar.\_\_new\_\_ is used to create instance of foobar.

type.\_\_new\_\_ is used to create the Foobar class

How instance is created :

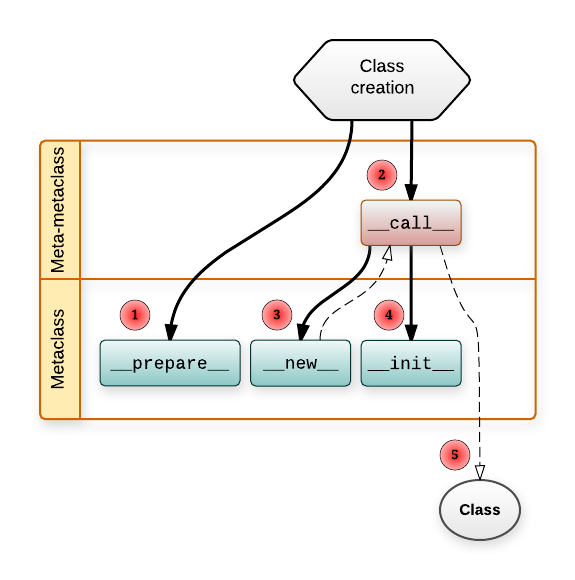


How to read this swim lane diagram:

* The horizontal lanes is the place where you define the functions.
* Solid lines mean a function call.
  + A line from Metaclass.\_\_call\_\_ to Class.\_\_new\_\_ means Metaclass.\_\_call\_\_ will call Class.\_\_new\_\_.
* Dashed lines means something is returned.
  + Class.\_\_new\_\_ returns the instance of Class.
  + Metaclass.\_\_call\_\_ returns whatever Class.\_\_new\_\_ returned (and if it returned an instance of Class it will also call Class.\_\_init\_\_ on it). [[16]](https://blog.ionelmc.ro/2015/02/09/understanding-python-metaclasses/#new-and-init)
* The number in the red circle signifies the call order.

=========================================================================

Creating a class is quite similar:



Few more notes:

* Metaclass.\_\_prepare\_\_ just returns the namespace object (a dictionary-like object).
* Metaclass.\_\_new\_\_ returns the Class object.
* MetaMetaclass.\_\_call\_\_ returns whatever Metaclass.\_\_new\_\_ returned (and if it returned an instance of Metaclass it will also call Metaclass.\_\_init\_\_ on it).