- How indexing works in sets
- Why dict key cant be mutable data types
- Enumerate
- destructor
- dir/isinstance/issubclass
- classmethod vs staticmethod
- The diamond problem
- What’s the meaning of single and double underscores in Python variable and method names
- Magic Methods (repr vs str)
- How can objects be stored in sets even though they are mutable

```
s = {21,34,11,56,39}
s
```

```
{11, 21, 34, 39, 56}
```

```
d = {(1,2,3):'nitish'}
d
```

```
{(1, 2, 3): 'nitish'}
```

```
d = {[1,2,3]:'nitish'}
d
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-99-968dc6d378a3> in <module>
----> 1 d = {[1,2,3]:'nitish'}
      2 d

TypeError: unhashable type: 'list'
```

```
# enumerate
# The enumerate() method adds a counter to an iterable and returns it (the enumerate object).
L = [('nitish',45),('ankit',31),('ankita',40)]

sorted(L,key=lambda x:x[1],reverse=True)
```

```
[('nitish', 45), ('ankita', 40), ('ankit', 31)]
```

```
L = [15,21,13,13]
sorted(list(enumerate(L)),reverse=True)
```

```
[(3, 13), (2, 13), (1, 21), (0, 15)]
```

```
# destructor
class Example:

  def __init__(self):
    print('constructor called')

  # destructor
  def __del__(self):
    print('destructor called')

obj = Example()
a = obj
del obj
del a
```

```
constructor called
destructor called
```

```
# dir

class Test:
    def __init__(self):
        self.foo = 11
        self._bar = 23
        self.__baz = 23

    def greet(self):
      print('hello')

t = Test()
print(dir(t)) # This gives us a list with the object’s attributes
```

```
['_Test__baz', '__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt
```

```
# isinstance

class Example:

  def __init__(self):
    print('hello')

obj = Example()

isinstance(obj,Example)
```

```
hello
True
```

```
# issubclass
class A:
  def __init__(self):
    pass

class B(A):
  pass

issubclass(B,A)
```

```
True
```

## classmethod

- A class method is a method that is bound to the class and not the object of the class.
- They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance.
- It can modify a class state that would apply across all the instances of the class. For example, it can modify a class variable that will be applicable to all the instances.

## staticmethod

A static method does not receive an implicit first argument. A static method is also a method that is bound to the class and not the object of the class. This method can’t access or modify the class state. It is present in a class because it makes sense for the method to be present in class.

```
class A:

  def normal_m(self):
    print('normal method')

  @staticmethod
  def static_m():
    print('static method')

  @classmethod
  def class_m(cls):
    print('class method')
```

```
a = A()

# normal -> object -> callable
a.normal_m()
# class -> object -> callable
a.class_m()
# static -> object -> not callable
a.static_m()
```

```
normal method
class method
static method
```

```
# static -> class -> callable
A.static_m()
# class method -> class -> callable
A.class_m()
# normal -> class -> not callable
A.normal_m()
```

```
static method
class method
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-94-088d25a52aaf> in <module>
      4 A.class_m()
      5 # normal -> class -> not callable
----> 6 A.normal_m()

TypeError: normal_m() missing 1 required positional argument: 'self'
```

```
# Alternate syntax
A.normal_m(a)
```

## Class method vs Static Method

The difference between the Class method and the static method is:

- A class method takes cls as the first parameter while a static method needs no specific parameters.
- A class method can access or modify the class state while a static method can’t access or modify it.
- In general, static methods know nothing about the class state. They are utility-type methods that take some parameters and work upon those parameters. On the other hand class methods must have class as a parameter.
- We use @classmethod decorator in python to create a class method and we use @staticmethod decorator to create a static method in python.

## When to use the class or static method?

- We generally use the class method to create factory methods. Factory methods return class objects ( similar to a constructor ) for different use cases.
- We generally use static methods to create utility functions.

```
# The diamond problem
class Class1:
    def m(self):
        print("In Class1")

class Class2(Class1):
    def m(self):
        print("In Class2")

class Class3(Class1):
    def m(self):
        print("In Class3")

class Class4(Class3, Class2):
    pass

obj = Class4()
obj.m()
# MRO
```

```
In Class3
```

```
# Double and single score
```

```
# repr and other magic/dunder methods

a = 'hello'

print(str(a))
print(repr(a))
```

```
hello
'hello'
```

```
import datetime

a = datetime.datetime.now()
b = str(a)

print(str(a))
print(str(b))

print(repr(a))
print(repr(b))
```

```
2022-11-26 15:46:52.007475
2022-11-26 15:46:52.007475
datetime.datetime(2022, 11, 26, 15, 46, 52, 7475)
'2022-11-26 15:46:52.007475'
```

## In summary

- str is for users -> meant to be more readable
- repr is for developers for debugging - > for being unambigous

```
# how objects are stored even though they are mutable
# https://stackoverflow.com/questions/31340756/python-why-can-i-put-mutable-object-in-a-dict-or-set
class A:

  def __init__(self):
    print('constructor')

  def hello(self):
    print('hello')

a = A()
a.hello()
s = {a}
print(s)
dir(a)
```

```
constructor
hello
```

```
{<__main__.A object at 0x7f4f5f3fd510>}
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'hello']
```

```
class A:

  def __init__(self):
    print('constructor')

  def __eq__(self):
    pass

  def __hash__(self):
    return 1

  def hello(self):
    print('hello')

a = A()
a.hello()
s = {a}
print(s)

dir(a)
```

```
constructor
hello
{<__main__.A object at 0x7f4f5f369290>}
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'hello']
```

```
class A:

  def __init__(self):
    self._var = 10

a = A()
a._var
```

```
10
```

```
s = {[1,2]}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-125-abf442ad56c0> in <module>
----> 1 s = {[1,2]}

TypeError: unhashable type: 'list'
```

```
L = [1,2,3]
s = {L}
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-129-fc3c139945bb> in <module>
      1 L = [1,2,3]
----> 2 s = {L}

TypeError: unhashable type: 'list'
```

```
print(L.__hash__)
```

```
None
```

```
hash(1)
```

```
1
```

```
hash('hello')
```

```
4306082800328210013
```

```
hash((1,2,3,))
```

```
2528502973977326415
```

```
hash([1,2,3])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-141-35e31e935e9e> in <module>
----> 1 hash([1,2,3])

TypeError: unhashable type: 'list'
```

Start coding or generate with AI.