Python_advance_assignment_4

Q1. Which two operator overloading methods can you use in your classes to support iteration?

In [ ]:

```
Ans: __iter__ and __next__ are the operator overloading methods in python that support
iteration and are collectively called iterator protocol.
1> __iter__ returns the iterator object and is called at the start of loop in our
respective class.
2> __next__ is called at each loop increment, it returns the incremented value.
Also Stopiteration is raised when there is no value to return.
```

In [37]:

```
class Counter:
def __init__(self,low,high):
        self.current =low
        self.high =high
def __iter__(self):
        return self
def __next__(self):
        if self.current > self.high:
            raise StopIteration
        else:
            self.current += 1
            return self.current - 1
for ele in Counter(5,15):
         print(ele, end=" ")
```

```
  File "<ipython-input-37-dbce01c7035a>", line 2
    def __init__'(self,low,high)':
    ^
IndentationError: expected an indented block
```

Q2. In what contexts do the two operator overloading methods manage printing?

In [ ]:

```
Ans: __str__ and __repr__ are two operator overloading methods that manage printing.

In Short, the difference between both these operators is the goal of __repr__is to be
unambiguous and __str__is to be readable.

Whenever we are printing any object reference internally __str__ method will be called
by default.

The main purpose of __str__ is for readability. it prints the informal string
representation of an object, one that is useful for printing the object. it may not be
possible to convert result string to original object.

__repr__ is used to print official string representation of an object,so it includes
all **Ans:** informationanddevelopment.
```

In [16]:

```
class Student :
    def __init__(self,name,roll_no):
        self.name = name
        self.roll_no = roll_no

s1 = Student("Mano",1)
print(str(s1))

class Student:
    def __init__(self,name,roll_no):
        self.name = name
        self.roll_no = roll_no
    def __str__(self):
        return f'Student Name:{self.name} and Roll No:{self.roll_no}'

s1 = Student("Mano",1)
print(str(s1))

import datetime
today = datetime.datetime.now()

s = str(today) # converting datetime object to presentable str
print(s)
try:d = eval(s) # converting str back to datetime object
except: print("Unable to convert back to original object")

u = repr(today) # converting datetime object to str
print(u)
e = eval(u) # converting str back to datetime object
print(e)
```

```
<__main__.Student object at 0x00000243EB458E20>
Student Name:Mano and Roll No:1
2023-07-06 08:56:55.170836
Unable to convert back to original object
datetime.datetime(2023, 7, 6, 8, 56, 55, 170836)
2023-07-06 08:56:55.170836
```

Q3. In a class, how do you intercept slice operations?

In [ ]:

```
Ans: In a class use of slice() in __getitem__ method is used for intercept slice
operation.
This slice method is provided with start integer number, stop integer number and step
integer number.
Example: __getitem__(slice(start,stop,step))
```

Q4. In a class, how do you capture in-place addition?

In [ ]:

```
Ans: a+b is normal addition. Whereas a += b is inplace addition operation.
In this in-place addition a itself will store the value of addition.
In a class __iadd__ method is used for this in-place operation.
```

In [30]:

```
class Book:
    def __init__(self,pages):
        self.pages = pages
    def __iadd__(self,other):
        self.pages += other.pages
        return self.pages
b1 = Book(700)
b2 = Book(200)
b1 += b2
print(b1)
```

```
900
```

Q5. When is it appropriate to use operator overloading?

In [ ]:

```
Ans: Operator overloading is used when we want to use an operator other than its
normal operation to have different meaning according to the context required in
user defined function.
```

In [39]:

```
class Book:
    def __init__(self,pages):
        self.pages = pages
        def __add__(self,other):
            return  self.pages+other.pages
b1 = 'Book(700)'
b2 = 'Book(200)'
print(f'Total Number of Pages ->{b1+b2}')
```

```
Total Number of Pages ->Book(700)Book(200)
```

In [ ]: