Assignment 04 Solutions

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

Ans: `__iter__` and `__next__` are the operator overloading methods in python that support iteration and are collectively called iterator protocol.

- `__iter__` returns the iterator object and is called at the start of loop in our respective class.
- `__next__` is called at each loop increment, it returns the incremented value. Also Stopiteration is raised when there is no value to return.

In [1]:

```python
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(8,82):
    print(ele, end=" ")
```

```
8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 4
8 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82
```

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

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

- The difference between both of 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 information and development.

In [2]:

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

s1 = Student("yono",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("bhavesh",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 0x00000256DF1472E0>
Student Name: bhavesh and Roll No: 1
2023-01-23 22:31:42.923429
Unable to convert back to original object
datetime.datetime(2023, 1, 23, 22, 31, 42, 923429)
2023-01-23 22:31:42.923429
```

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

Ans: In a class use of `slice()` , `__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))`

In [3]:

```python
sliced ='abhishek raju kale'.__getitem__(slice(0, 6, 1)) #using slice() and __getitem__ slicing can be achieved in class
print(sliced)
```

```
abhish
```

In [4]:

```python
class Demo:
    def __getitem__(self, key):
        # print a[1], a[1, 2],
        # a[1, 2, 3]
        print(key)
        #return key

a = Demo()
a[1]
a[1, 2]
a[1, 2, 3]
```

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

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

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 [5]:

```python
class Book:
    def __init__(self,pages):
        self.pages = pages
    def __iadd__(self,other):
        self.pages += other.pages
        return self.pages

b1 = Book(300)
b2 = Book(900)
b1+=b2
print(b1)
```

```
1200
```

Q5. When is it appropriate to use operator overloading?

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 [6]:

```python
class Book:
    def __init__(self,pages):
        self.pages = pages
    def __add__(self,other):
        return self.pages+other.pages
b1 = Book(600)
b2 = Book(300)
print(f'Total Number of Pages -> {b1+b2}')
```

```
Total Number of Pages -> 900
```