# Binary Tree - Assignments

In this assignment, we will store a list of students in a Binary Tree structure .

## 1. Define Class `Student`

For the `Student` class to support binary tree structure, derive `Student` class from `Node` class. Define class `Student` with following requirements.

- `Student` has another 2 instance attributes `first_name` and `last_name` .
- Implement its `__init__()` function to initialize its 2 attributes.
- Implement its `__str__()` function to print string in the format of `Student('Alan', 'Goh')` .
- Implement a `fullname` property which returns students full name in `first_name last_name` format.

In [21]:

```
class Node:

    def __init__(self, data=None, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right

    def __str__(self):
        return '{}({},{})'.format(self.data,
                                  self.left.data if self.left else '',
                                  self.right.data if self.right else '')

```

In [53]:

```
class Student(Node):

    def __init__(self, first_name, last_name, left=None, right=None):
        self.first_name = first_name
        self.last_name = last_name
        self.left = left
        self.right = right

    def __str__(self):
        return "{}('{}','{}')".format(self.__class__.__name__, self.first_name, self.la

    @property
    def fullname(self):
        return self.first_name + " " + self.last_name

```

In [20]:

```
class Student2:

    def __init__(self, first_name, last_name):
        self.first_name = first_name
        self.last_name = last_name

    def __str__(self):
        return "{}('{}','{}')".format(self.__class__.__name__, self.first_name, self.la


```

Test:

In [48]:

```
s = Student('Alan', 'Goh')
print(s.fullname)
```

```
Alan Goh
```

## 2. Tree of Students

Create following student objects and form them into a binary tree structure. Its root node, i.e. Alan Goh, is pointed by variable `root`.


In [56]:

```
# YOUR CODE HERE

hp = Student('Henry', 'Poh')
kb = Student('Kelly', 'Beh')
dm = Student('Denny', 'Mok')

pt = Student('Peter', 'Tan', hp, kb)
gc = Student('Genny', 'Chen', None, dm)

root = Student('Alan', 'Goh', pt, gc)

print(root)
```

```
Student('Alan','Goh')
```

Define a binary tree class `StudentTree` to supports printing of binary tree.

- You can derive `StudentTree` from `BinaryTree` class.
- Such tree will print above student tree as following:

```
Student('Alan', 'Goh')
Student('Peter', 'Tan') Student('Genny', 'Chen')
Student('Henry', 'Poh') Student('Kelly', 'Beh') Student('Denny', 'Mok')
```

In [36]:

```
class BinaryTree:

    def __init__(self, root=None):
        self.root = root

    def print_tree(self):
        self._print_tree([self.root])

    def _print_tree(self, node_list):
        # Convert node_list to a list if it is not
        if not isinstance(node_list, list):
            node_list = [node_list]
        # Stop recursion if the list is empty
        if not node_list:
            return
        # define a list to collect nodes in next layer
        next_layer = []
        while node_list:
            node = node_list.pop()
            print(node, end=' ')
            if node.left:
                next_layer.insert(0, node.left)
            if node.right:
                next_layer.insert(0, node.right)
        print()
        self._print_tree(next_layer)
```

In [37]:

```
class StudentTree(BinaryTree):
    pass

```

Test:

In [38]:

```
students = StudentTree(root)
students.print_tree()
```

```
Student2('Alan','Goh')(Student2('Peter','Tan'),Student2('Genny','Chen'))
Student2('Peter','Tan')(Student2('Henry','Poh'),Student2('Kelly','Beh')) Stu
dent2('Genny','Chen')(,Student2('Denny','Mok'))
Student2('Henry','Poh')(,) Student2('Kelly','Beh')(,) Student2('Denny','Mo
k')(,)
```

### Version 2

In [29]:

```
hp = Node(Student2('Henry', 'Poh'))
kb = Node(Student2('Kelly', 'Beh'))
dm = Node(Student2('Denny', 'Mok'))

pt = Node(Student2('Peter', 'Tan'), hp, kb)
gc = Node(Student2('Genny', 'Chen'), right=dm)

root = Node(Student2('Alan', 'Goh'), pt, gc)

print(root.data)
```

```
Student2('Alan','Goh')
```

In [33]:

```
class BinaryTree:

    def __init__(self, root=None):
        self.root = root

    def print_tree(self):
        self._print_tree([self.root])

    def _print_tree(self, node_list):
        # Convert node_list to a list if it is not
        if not isinstance(node_list, list):
            node_list = [node_list]
        # Stop recursion if the list is empty
        if not node_list:
            return
        # define a list to collect nodes in next layer
        next_layer = []
        while node_list:
            node = node_list.pop()
            print(node.data, end=' ')
            if node.left:
                next_layer.insert(0, node.left)
            if node.right:
                next_layer.insert(0, node.right)
        print()
        self._print_tree(next_layer)

```

In [34]:

```
class StudentTree(BinaryTree):
    pass
```

In [35]:

```
students = StudentTree(root)
students.print_tree()
```

```
Student2('Alan','Goh')
Student2('Peter','Tan') Student2('Genny','Chen')
Student2('Henry','Poh') Student2('Kelly','Beh') Student2('Denny','Mok')
```

## 3. Traverse the Tree in Pre-Order

Define an enhanced version of the `StudentTree` class by adding a `preorder()` instance function.

- The `preorder()` function traverse through the tree and print out student's fullname in each node.
- The printout is of following format for above tree.

```
Alan Goh; Peter Tan; Henry Poh; Kelly Beh; Genny Chen; Denny Mok;
```

In [61]:

```
class StudentTree(BinaryTree):

    def preorder(self):
        self._preorder(self.root)

    def _preorder(self, node=None):
        if node is None:
            return

        print(node.fullname, end='; ')
        self._preorder(node.left)
        self._preorder(node.right)


```

Tree:

In [62]:

```
students = StudentTree(root)
students.preorder()
```

```
Alan Goh; Peter Tan; Henry Poh; Kelly Beh; Genny Chen; Denny Mok;
```