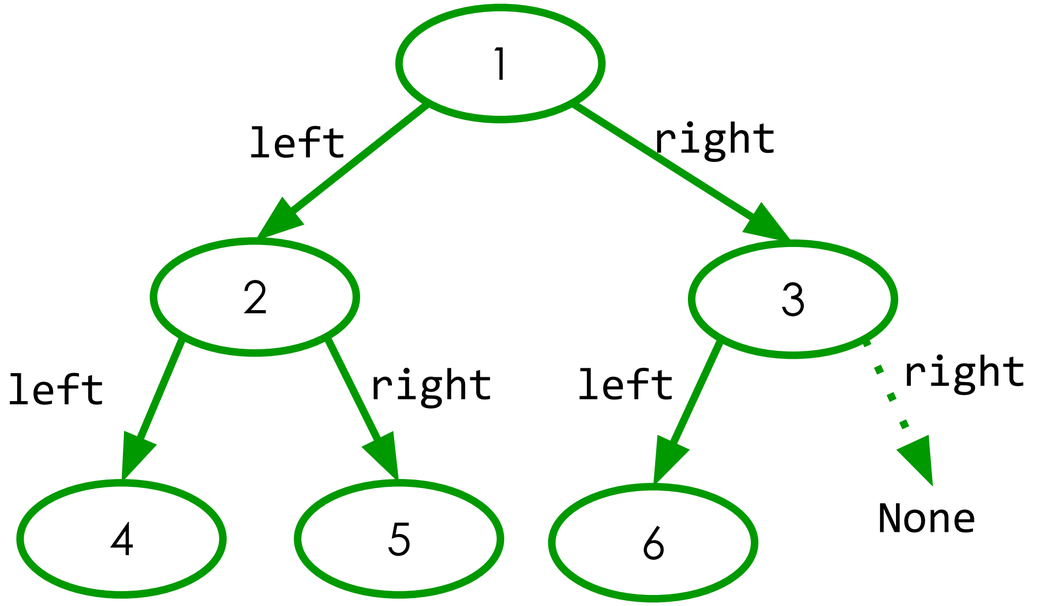
② 二分探索木の実装

Pythonで二分探索木を実装する手順を学習します。

（復習）二分木の実現方法

- 二分木を実装するには、リンクリストと同様に、各ノードから他のノードへのリンク（メモリアドレスなど）をもたせます
- 二分木とするために、以下のノードへのポイントを持たせる方法が一般的です
  - 左の子ノード
  - 右の子ノード
  - ※親ノードへのリンクを持たせることもあります
- 子ノードがない場合はNoneとします



二分探索木の実装

- 二分探索木も、二分木と同様のデータ構造で実装することができます
  - また二分木の中身を確認するために、通りがけ順の探索を行う関数も用意しておきましょう

```
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

    def __str__(self):
        return str(self.value)
```

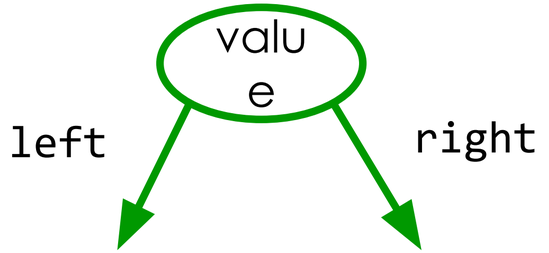
value：このノードの格納する要素
left：左の子ノード
right：右の子ノード

str に型変換した時はノードの値とする

```
def print_inorder(node):
    if node.left != None:
        print_inorder(node.left)

    print(node, end=', ')

    if node.right != None:
        print_inorder(node.right)
```



要素を追加するには？

- ノード `node` と値 `value` を引数にとり、二分探索木に要素を追加する関数 `add(node, value)` を定義します
  - ノードを返すようにすると、再帰を使って以下のように定義できます

```
def add(node, value):
    if node == None:
        return Node(value)

    if node.value == value:
        raise Exception('Already added.')

    if value < node.value:
        node.left = add(node.left, value)
        return node

    if value > node.value:
        node.right = add(node.right, value)
        return node
```

ノードがNoneの場合は、ノードを生成して返しします

値が一致した場合は追加できないため、例外を発生させます

追加する値の方がノードの値より小さい場合は、左の子を対象に再帰呼出しを行います
返り値を左の子とします（このことで、子がNoneだった場合に新たに生成されたノードが追加されます）

追加する値の方がノードの値より大きい場合は、同様に右の子を対象に再帰呼出しを行います

動作を確認してみましょう

- 以下のように空の二分探索木に要素を追加し、中身を確認しましょう
  - 通りがけ順の探索を行えば、昇順に表示されますね

```
root = None
root = add(root, 3)
root = add(root, 5)
root = add(root, 2)
root = add(root, 1)
root = add(root, 10)
root = add(root, 8)

print_inorder(root)
```

```
1, 2, 3, 5, 8, 10,
```

```
root = add(root, 9)
root = add(root, 4)

print_inorder(root)
```

```
1, 2, 3, 4, 5, 8, 9, 10,
```

要素を検索するには？

- ノード `node` と値 `value` を引数にとり、二分探索木のノードを検索する関数 `contains(node, value)` を定義します
  - 含まれていた場合は `True` を、そうでない場合は `False` を返します

```
def contains(node, value):
    if node == None:
        return False

    if node.value == value:
        return True

    if value < node.value:
        return contains(node.left, value)

    if value > node.value:
        return contains(node.right, value)
```

ノードがNoneの場合は、含まれていません

値が一致した場合は含まれています

追加する値の方がノードの値より小さい場合は、左の子を対象に再帰呼出しを行います

追加する値の方がノードの値より大きい場合は、右の子を対象に再帰呼出しを行います

動作を確認してみましょう

- 先ほど作成した二分探索木で、動作を確認しましょう

```
contains(root, 10)
```

```
True
```

```
contains(root, 7)
```

```
False
```

（参考）要素を削除するには？

- ノード `node` と値 `value` を引数にとり、二分探索木の要素を削除する関数 `delete(node, value)` を定義します
  - ここでもノードを返すようにします（根のノードが変わることもあるので注意！）

```
def delete(node, value):
    if node == None:
        raise Exception('Not added.')

    if node.value == value:
        if node.left != None and node.right != None:
            (right_node, min_node) = delete_min(node.right)
            node.right = right_node
            node.value = min_node.value
            return node
        if node.left != None:
            return node.left
        if node.right != None:
            return node.right
        return None
```

```
    if value < node.value:
```

ノードがNoneの場合は、削除できません

ノードと値が一致した場合…

子を2つ持つ場合は、右の子を根とする部分木の最小ノードを削除し、ノードの値をその値で置き換えます

子を1つ持つ場合は、子ノードを格上げします

子を持たない場合は、ノードを削除します

（参考）要素を削除するには？

- ノード `node` と値 `value` を引数にとり、二分探索木の要素を削除する関数 `delete(node, value)` を定義します
  - ここでもノードを返すようにします（根のノードが変わることもあるので注意！）

```
def del
    if
                                    if value < node.value:
                                        node.left = delete(node.left, value)
                                        return node

    if
                                    if value > node.value:
                                        node.right = delete(node.right, value)
                                        return node
        node.right = right_node
        node.value = min_node.value
        return node
    if node.left != None:
        return node.left
    if node.right != None:
        return node.right
    return None
```

削除する値の方がノードの値より小さい場合は、左の子を対象に再帰呼出しを行います

削除する値の方がノードの値より大きい場合は、右の子を対象に再帰呼出しを行います

（参考）要素を削除するには？

- なお「木の最小ノードを削除する」処理は、以下のように記述します
  - ここでは、削除した後の根ノードと、削除されたノードのタプルを返しています

```
def delete_min(node):
    if node.left != None:
        (left_node, min_node) = delete_min(node.left)
        node.left = left_node
        return (node, min_node)

    return (node.right, node)
```

左の子がいる場合は、左の子を再帰的に探索します

左の子がいない場合は、このノードが削除対象です