Tree Traversal

> Attempt to visit all the nodes in a tree.

Use a function `visitNode()` to make something to the value of a node (like printing it) and a recursive function `recurse()` to actually go from one node to another.

> Important
>
> Depending on where the function `visitNode()` is located, we are going to get a different type of traversal (pre, in or post).

Pre-Order

Putting `visitNode()` **before** the recursion. It visits in the order:

1. Node
2. Left Node
3. Right Node

```
def traversal_pre(root, path):
        if root.value == None:
                return path

        path.append(node.value) # visitNode() !!!
        travesal(root.left, path)
        traversal(root.right, path)

        return path

root = ROOT
path = []
def main():
        traversal(root, path)
        print(path)
```

In-Order

Putting `visitNode()` **in between** both recursion calls. It visits in the order:

1. Left Node
2. Node
3. Right Node

```
def traversal(root, path):
        if root.value == None:
                return path

        travesal(root.left, path)
        path.append(node.value) # visitNode() !!!
        traversal(root.right, path)

        return path

root = ROOT
path = []
def main():
        traversal(root, path)
        print(path)
```

Pos-Order

Putting `visitNode()` **after** both recursion calls. It visits in the order:

1. Left Node
2. Right Node

3. Node

```
def traversal(root, path):
        if root.value == None:
                return path

        travesal(root.left, path)
        traversal(root.right, path)
        path.append(node.value) # visitNode() !!!

        return path

root = ROOT
path = []
def main():
        traversal(root, path)
        print(path)
```

It just copy-pasted code, so simple!

In C

```c
#include <stdio.h>
#include <stdlib.h>

// Define the node structure for the binary tree
typedef struct Node {
    int value;
    struct Node* left;
    struct Node* right;
} Node;

// Define the path structure
typedef struct {
    int* data;
    int size;
    int capacity;
} Path;

// Function to create a new node
Node* createNode(int value) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->value = value;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

// Function to initialize the path structure
Path* createPath() {
    Path* path = (Path*)malloc(sizeof(Path));
    path->size = 0;
    path->capacity = 10; // Initial capacity
    path->data = (int*)malloc(path->capacity * sizeof(int));
    return path;
}

// Function to append a value to the path
void appendToPath(Path* path, int value) {
    if (path->size >= path->capacity) {
        path->capacity *= 2;
        path->data = (int*)realloc(path->data, path->capacity * sizeof(int));
    }
    path->data[path->size++] = value;
}

// Function to perform in-order traversal of the binary tree
void traversal(Node* root, Path* path) {
    if (root == NULL) {
        return;
    }

    appendToPath(path, root->value); // visitNode()
    traversal(root->left, path);
    traversal(root->right, path);
}

// Function to print the path
void printPath(Path* path) {
    for (int i = 0; i < path->size; i++) {
        printf("%d ", path->data[i]);
    }
    printf("\n");
}

// Main function
int main() {
    // Creating a simple binary tree for demonstration
    Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->left = createNode(6);
    root->right->right = createNode(7);

    Path* path = createPath();
    traversal(root, path);
    printPath(path);

    // Freeing allocated memory
    free(path->data);
    free(path);

    // Freeing the tree nodes
    free(root->left->left);
    free(root->left->right);
    free(root->left);
    free(root->right);
    free(root);

    return 0;
}
```

Just change the location of `appendToPath` and see how the traversal is done.

Complete Python Code

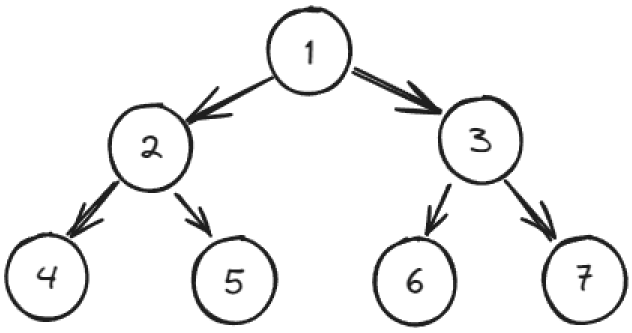
We still needed to implement the `Node` class, we need to use a class to substitute the C struct:

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

def traversal(root, path):
    if root is None:
        return path

    path.append(root.value) # visitNode() !!!
    traversal(root.left, path)
    traversal(root.right, path)

    return path

def main():
    root = Node(1)
    root.left = Node(2)
    root.right = Node(3)
    root.left.left = Node(4)
    root.left.right = Node(5)

    path = []
    traversal(root, path)
    print(path)
```

```
if __name__ == "__main__":
    main()
```

What happens to the root node?

In the following binary tree:



The resulting ordering in the 3 different cases is:

- Pre: `1, 2, 3, 4, 5, 6, 7`
- In: `4, 2, 5, 1, 3, 6, 7`
- Pos: `4, 5, 2, 6, 7, 3, 1`

The root is at the beginning, the middle and the end!!!

```
`**1**, 2, 3, 4, 5, 6, 7`
`4, 2, 5, **1**, 3, 6, 7`
`4, 5, 2, 6, 7, 3, **1**`
```

In behavior corresponds to all the binary trees.