Studying Notes: Data Structures and Algorithms in Python

Stack

Operations

Stack operations:

```
push()
pop()
peek()
is_empty()
```

- `push(value)` inserts elements onto the stack. The value being pushed becomes the new Top value
- `pop()` removes the element at top and returns its value
- `peek()` returns the value at the top of the stack. Note: the element at Top is NOT removed with this operation.
- `is_empty()` returns True if the stack is empty, otherwise it returns False

```
# Stack Data Structure

class Stack():
    def __init__(self):
        self.items = []

    def push(self, value):
        self.items.append(value)

    def pop(self):
        return self.items.pop()

    def peek(self):
        return self.items[len(self.items)-1]

    def is_empty(self):
        return len(self.items) == 0
```

```
# Demonstration
stack = Stack()
stack.push("A")
# [A]
stack.push("B")
```

```
# [A, B]
stack.push("C")
# [A, B, C]
stack.peek()
# C
stack.pop()
# [A, B], C is returned
stack.peek()
# B
stack.pop()
# [A]
stack.pop()
# []
```

Problem: Balanced Braces

Balanced: ([]) Unbalanced: (() Edge Case:))

Algorithm:

- Iterate through the characters of the string
- If we encounter an opening bracket, push it onto the stack
- if we encounter a closing bracket, pop off an element from the stack and match it with the closing bracket.
    - If it is an opening bracket and of the same type as the closing bracket, we conclude it is a successful match and move on.
    - If it is not a match then the set of brackets is not balanced
- If the stack is empty after we have iterated through the list, then it is balanced. If the stack isn't empty then the set is not balanced and we should return False.

```
from stack import Stack
def is_paren_balanced(paren_string):
    s = Stack()
    is_balanced = True
    index = 0

    while index < len(paren_string) and is_balanced:
        paren = paren_string[index]
        if paren in "({[":
            s.push(paren)
        else:
            if s.is_empty():
                is_balanced = False
                break
            else:
                top = s.pop()
                if not is_match(top, paren):
                    is_balanced = False
                    break
```

```
            index += 1
        if s.is_empty() and is_balanced:
            return True
        else:
            return False


def is_match(p1, p2):
    if p1 == "(" and p2 == ")":
        return True
    elif p1 == "[" and p2 == "]":
        return True
    elif p1 == "{" and p2 == "}":
        return True
    else:
        return False
```

Problem: Reversing a string

Algorithm:

- Push all the characters onto the stack
- Pop all the characters off of the stack, into the passed list
- return the list

```
from stack import Stack
def reverse(s: Stack, string: List[str]) -> List[str]:
    for i in range(len(input_str)):
        stack.push(input_str[i])
    reverse_string = ""
    while not stack.is_empty():
        reverse_string += s.pop()
    return reverse_string
```

Problem: Convert Decimal Integer to Binary using Divide by Two

```
# Divide by two, extract the non-fractional portion of the answer and
record the division remainder
242 / 2: Value = 121, Remainder = 0 <- LSB
# Divide the previous result by two until you reach zero and continue
recording the remainders
121 / 2: Value = 60, Remainder = 1
60 / 2: Value = 30, Remainder = 0
30 / 2 : Value = 15, Remainder = 0
15 / 2: Value = 7, Remainder = 1
7 / 2: Value = 3, Remainder = 1
3 / 2: Value = 1, Remainder = 1
1 / 2: Value = 0, Remainder = 1 <- MSB
```

```
# Read from the bottom of the remainders (Most Significant Bit) to the top
(Least Significant Bit) to get the binary equivalent of the integer.
Binary Equivalent = 1111 0010
```

Operation	Value	Remainder	Bit Importance
242/2	121	0	LSB
121/2	60	1	
60/2	30	0	
30/2	15	0	
15/2	7	1	
7/2	3	1	
3/2	1	1	
1/2	0	1	MSB

Result: 1111 0010

```python
# Code example
from stack import Stack

def convert_int_to_bin(dec_num):
    # Intended code
    result = ""
    if dec_num:
        s = Stack()
        while dec_num > 0:
            # get the remainder
            s.push(str(dec_num % 2))
            dec_num = dec_num // 2
        while not s.is_empty():
            result += s.pop()
    return result

def convert_int_to_bin(dec_num):
    # Better approach: Eliminates use of stack altogether
    result = ""
    while dec_num > 0:
        # get the remainder
        result = str(dec_num % 2) + result
        dec_num = dec_num // 2
    if result == "":
        result = "0"
    return result
```