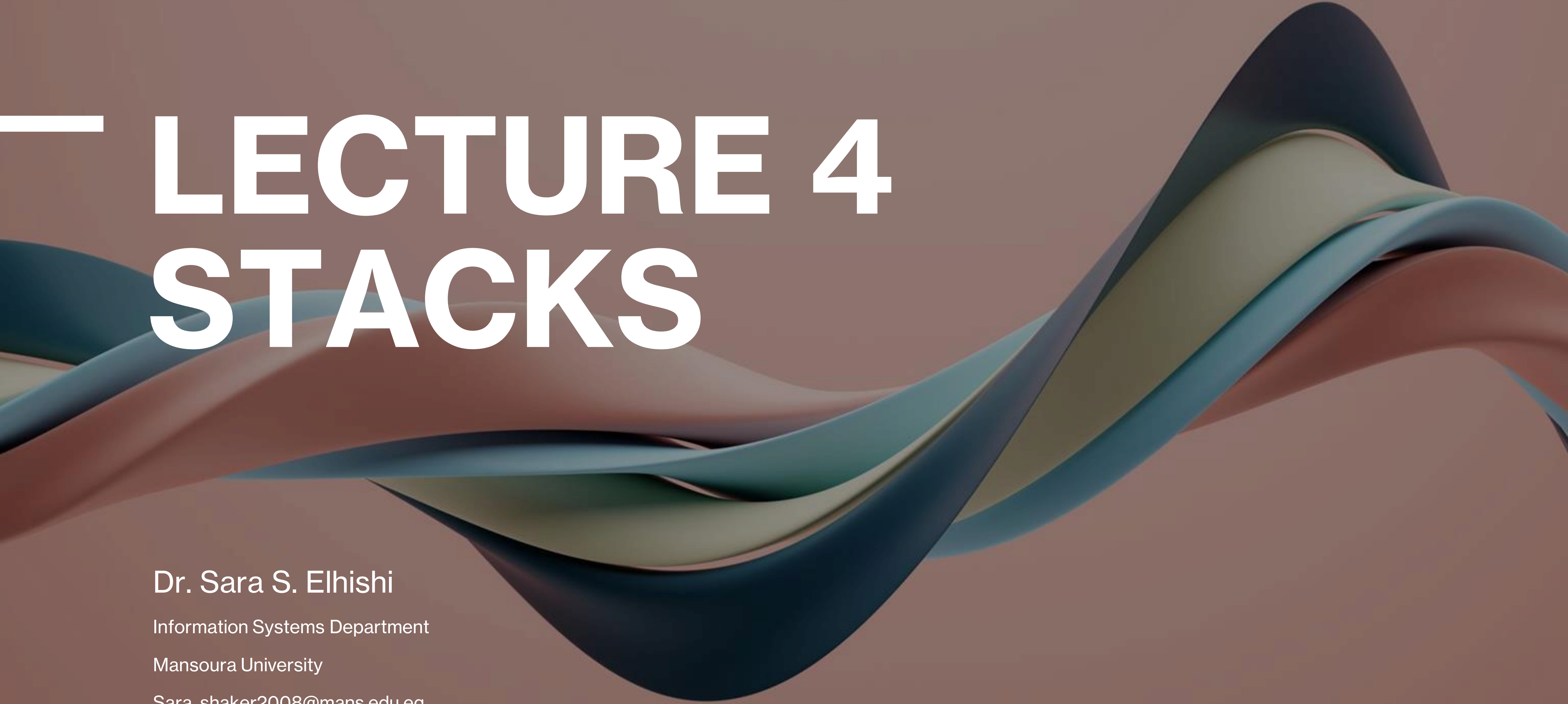


LECTURE 4
STACKS

Dr. Sara S. Elhishi

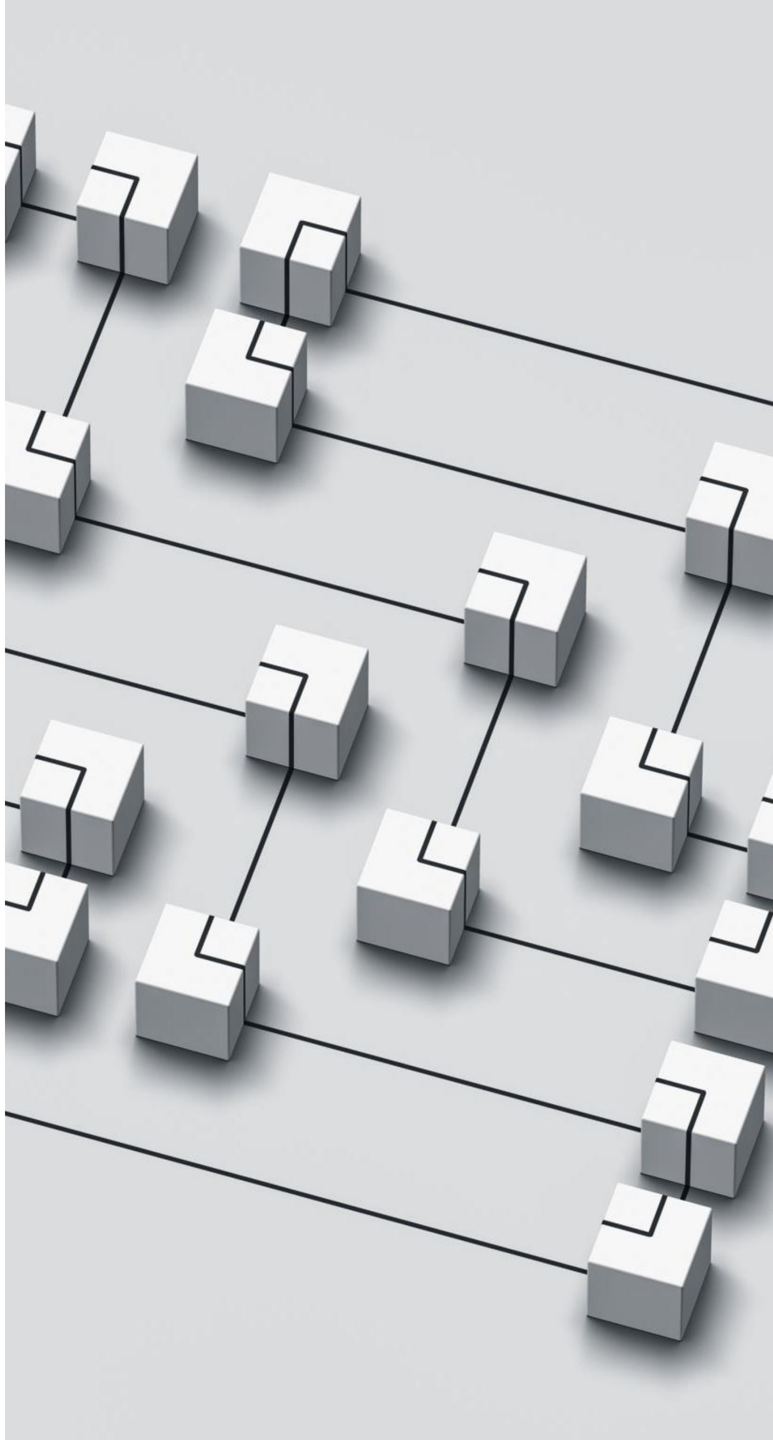
Information Systems Department

Mansoura University

Sara_shaker2008@mans.edu.eg

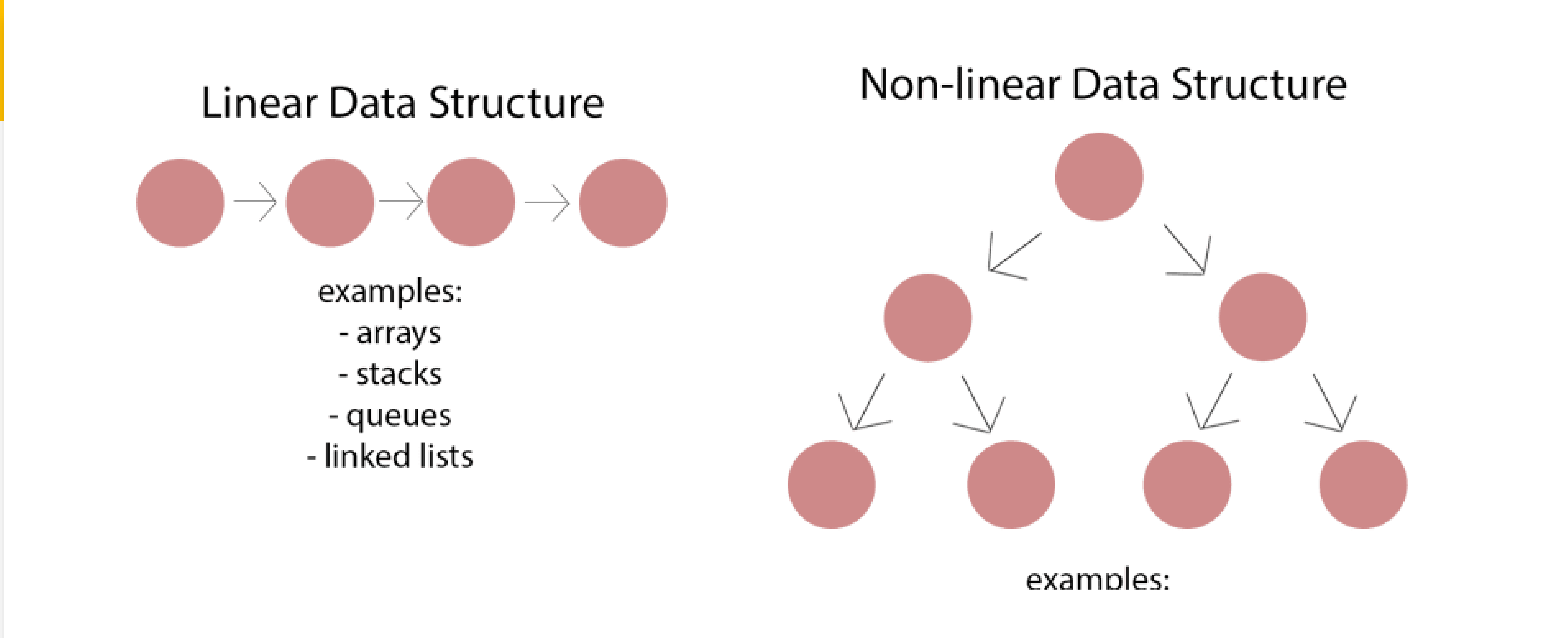
Data Structures

- Creating an effective algorithm is necessitated by including suitable data structures
- A data structure is a specialized format for organizing, processing, retrieving and storing data.
- Arrays, Lists, Stacks, Queues, Trees … etc.



Abstract Data Types (ADT)

- The data items that make up the data structure and their fundamental operations.
- ADT encourages data abstraction by emphasizing what a data structure does rather than how it does it.
- ADT is described as:
    - a collection of data items D that are
    - defined over a domain L and
    - support a number of operations O.
- Data structures task is defining **HOW** these operations are done (implemented)



Linear Data Structures

- Data Collections in which once an item is added, it stays in that position relative to the other elements that came before and came after it.
- Stacks, queues, deques, and lists are examples

The Appropriate Data Structure

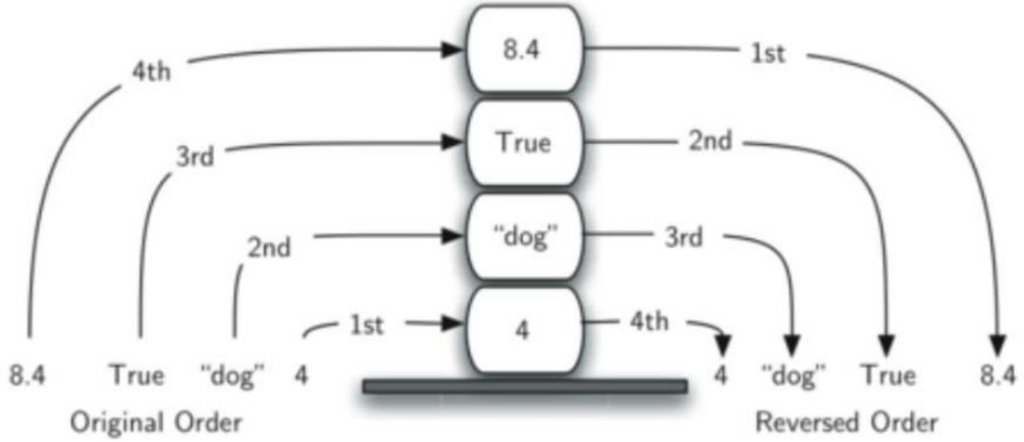
- Storage and Retrieval
- Restricted Access
- Abstract

Data Types and Abstraction

- Data Types
  - Primitive data types (e.g., int, float, ..)
- Abstract Data Type (ADT)
  - An abstraction captures the fundamental nature or significant attributes of something
  - ADT is a way of looking at a data structure – focusing on what it does and ignoring how it does its job.
  - Within the context of object-oriented programming, an abstract data type refers to a class
  - Interface

STACK

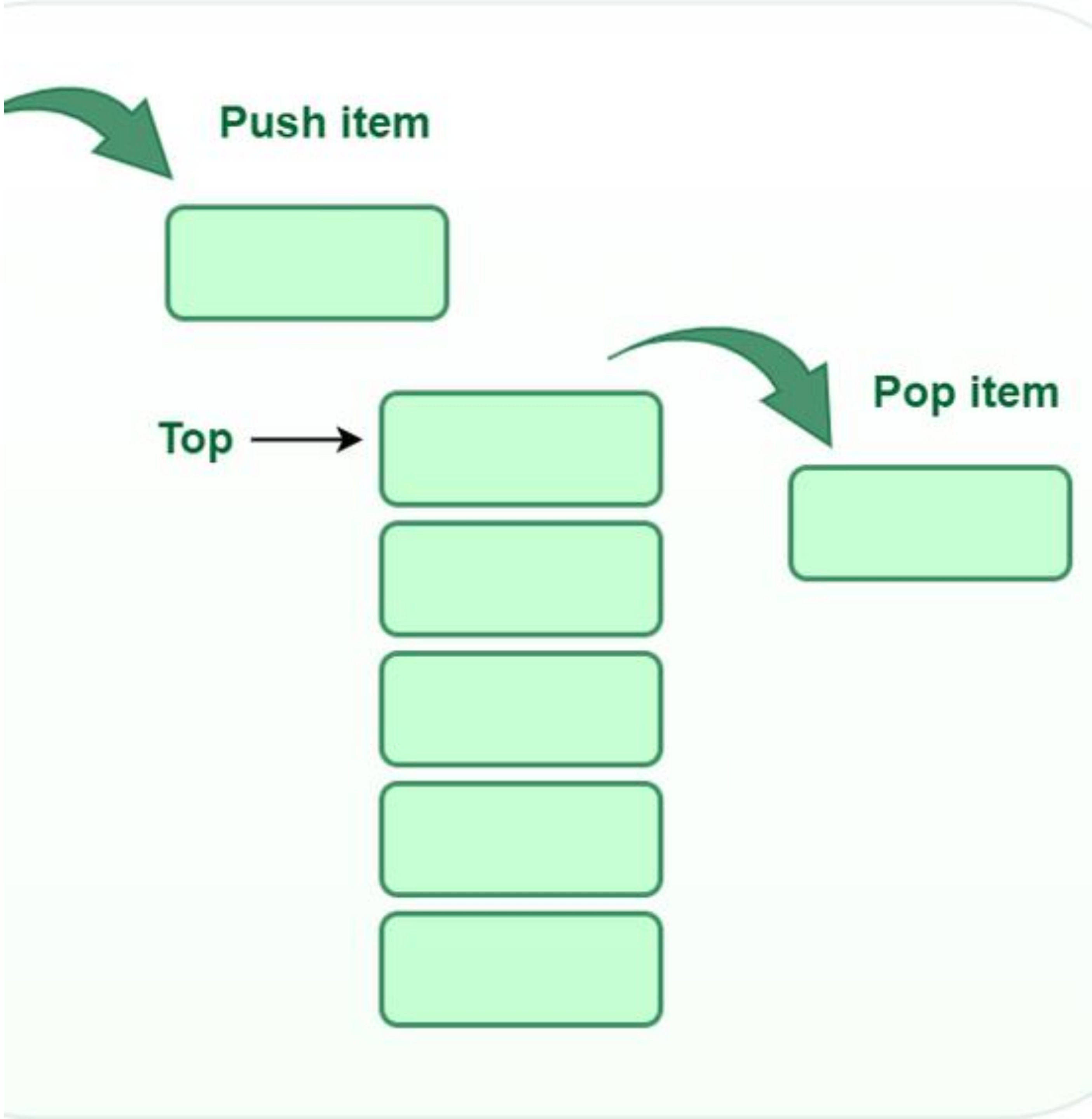
Stack



- A collection of objects inserted and removed according to the last-in, first-out (**LIFO**) principle.
- A stack data structure provides access to only the most recently inserted data item in the collection (**TOP**).
- By removing this item, you can access the item that was entered immediately before it, and so on.
- Browser history and Undo operations are examples of Stacks.

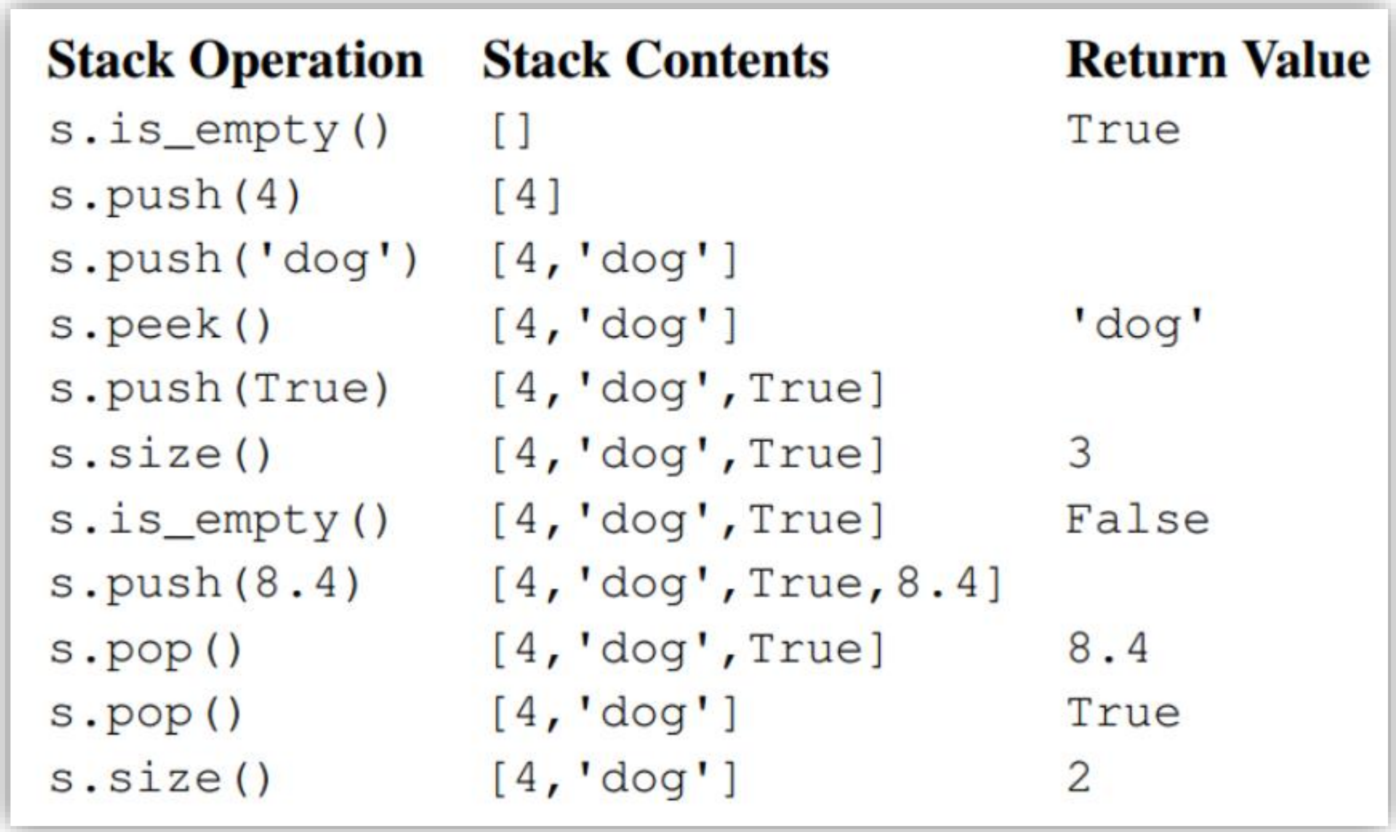
Stack Operations

- Push
- Pop
- Peek



Stack ADT

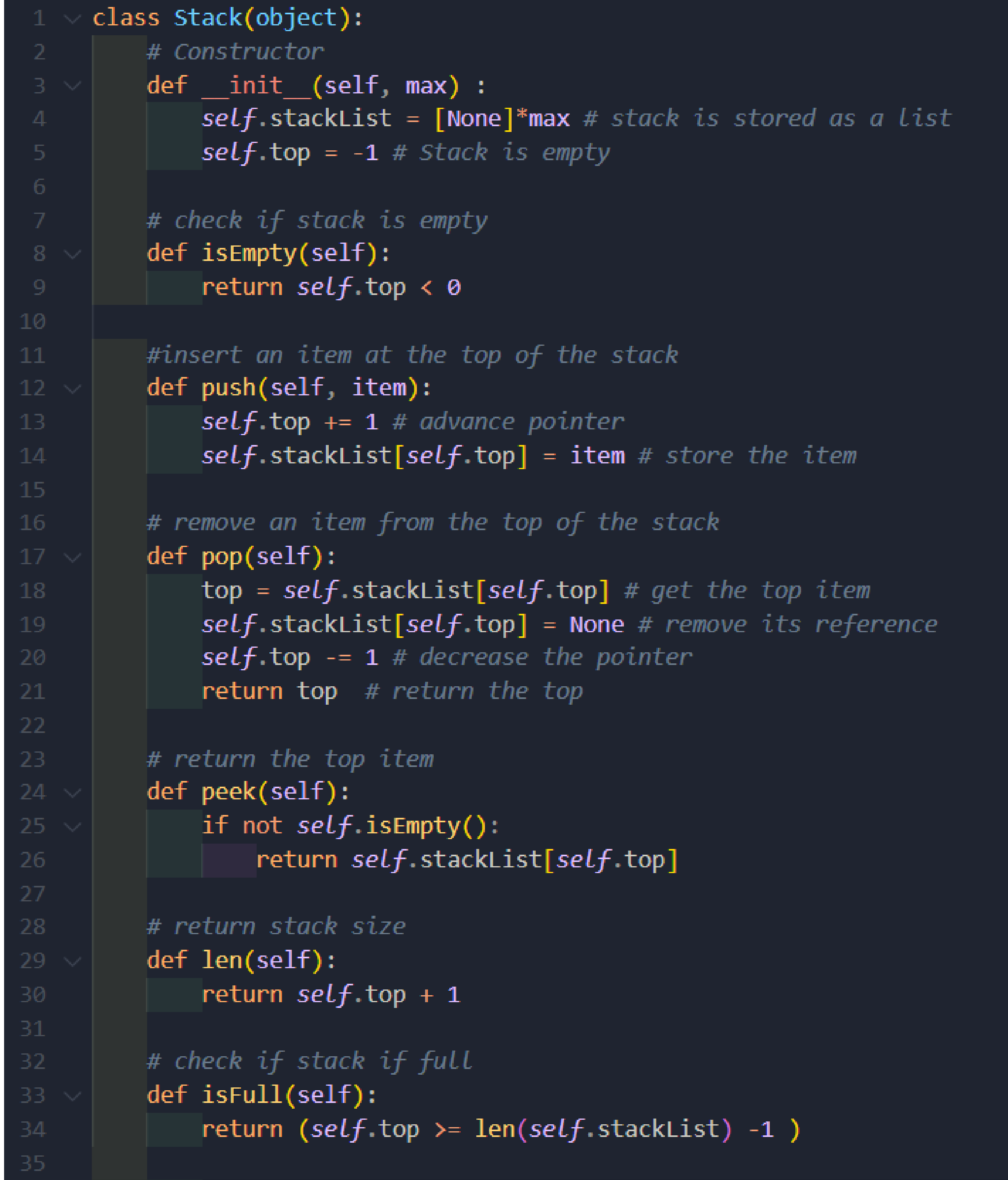
- stack()
- push()
- pop()
- peek()
- isempty()
- size()



Stack Operation	Stack Contents	Return Value
`s.is_empty()`	`[]`	`True`
`s.push(4)`	`[4]`	
`s.push('dog')`	`[4,'dog']`	
`s.peek()`	`[4,'dog']`	`'dog'`
`s.push(True)`	`[4,'dog',True]`	
`s.size()`	`[4,'dog',True]`	`3`
`s.is_empty()`	`[4,'dog',True]`	`False`
`s.push(8.4)`	`[4,'dog',True,8.4]`	
`s.pop()`	`[4,'dog',True]`	`8.4`
`s.pop()`	`[4,'dog']`	`True`
`s.size()`	`[4,'dog']`	`2`

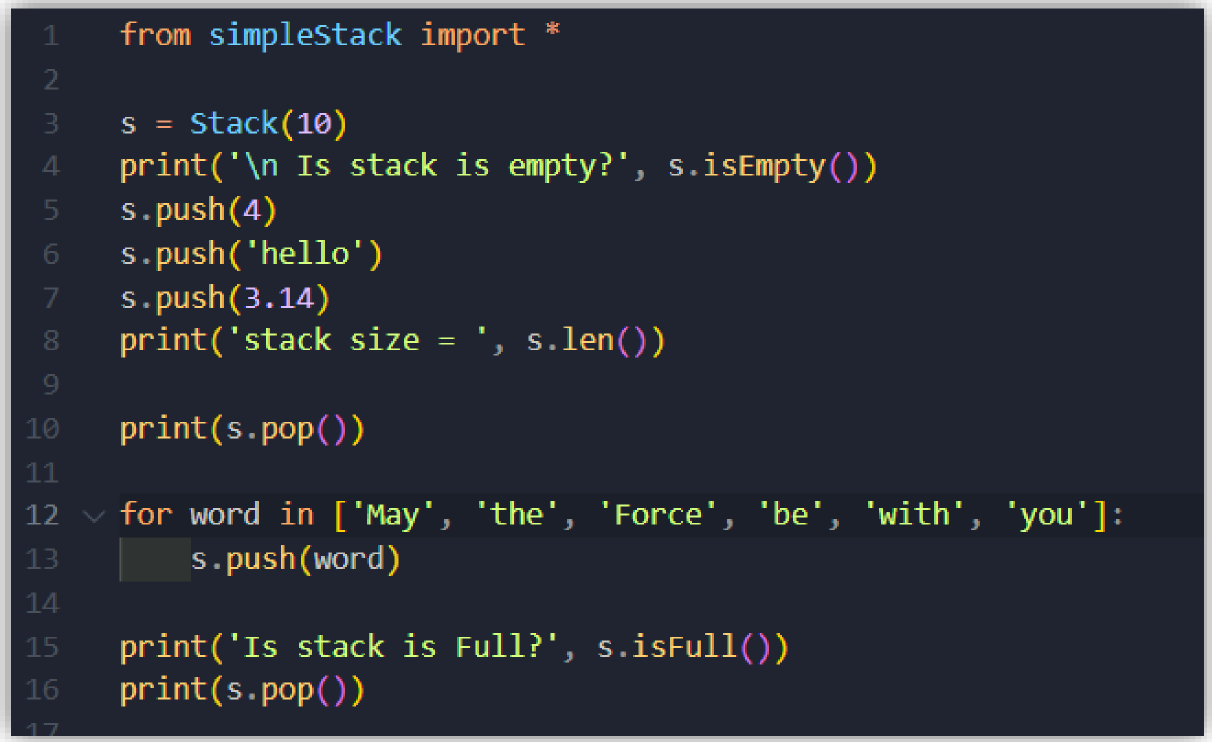
STACK IMPLEMENTATION

IMPLEMENTING A STACK IN PYTHON

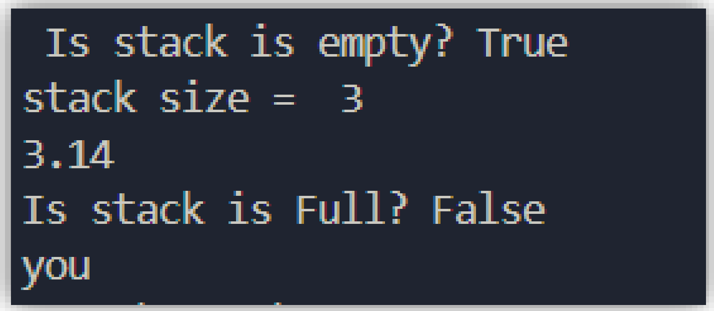


```python
class Stack(object):
    # Constructor
    def __init__(self, max) :
        self.stackList = [None]*max # stack is stored as a list
        self.top = -1 # Stack is empty

    # check if stack is empty
    def isEmpty(self):
        return self.top < 0

    #insert an item at the top of the stack
    def push(self, item):
        self.top += 1 # advance pointer
        self.stackList[self.top] = item # store the item

    # remove an item from the top of the stack
    def pop(self):
        top = self.stackList[self.top] # get the top item
        self.stackList[self.top] = None # remove its reference
        self.top -= 1 # decrease the pointer
        return top  # return the top

    # return the top item
    def peek(self):
        if not self.isEmpty():
            return self.stackList[self.top]

    # return stack size
    def len(self):
        return self.top + 1

    # check if stack if full
    def isFull(self):
        return (self.top >= len(self.stackList) -1 )

```

Stack Client Program



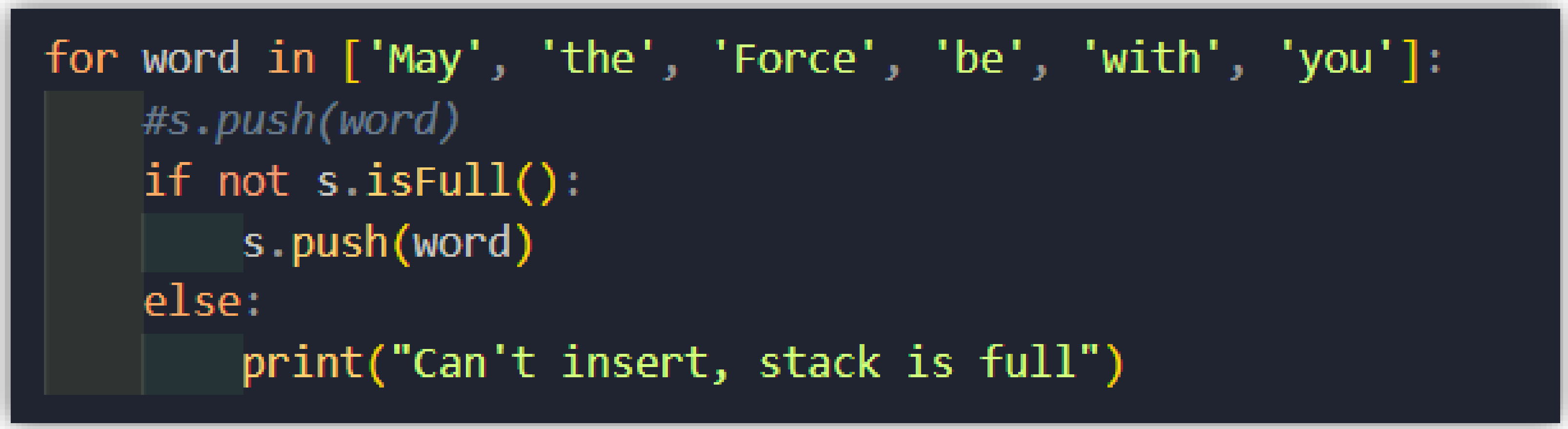
```python
from simpleStack import *

s = Stack(10)
print('\n Is stack is empty?', s.isEmpty())
s.push(4)
s.push('hello')
s.push(3.14)
print('stack size = ', s.len())

print(s.pop())

for word in ['May', 'the', 'Force', 'be', 'with', 'you']:
    s.push(word)

print('Is stack is Full?', s.isFull())
print(s.pop())
```



```
 Is stack is empty? True
stack size =  3
3.14
Is stack is Full? False
you
```

Error Handling

- Certain consequences should occur if you attempt to push an item onto a stack that is already full or pop an item from an empty stack.
- For example, The application should consistently verify the stack's capacity before inserting an item.



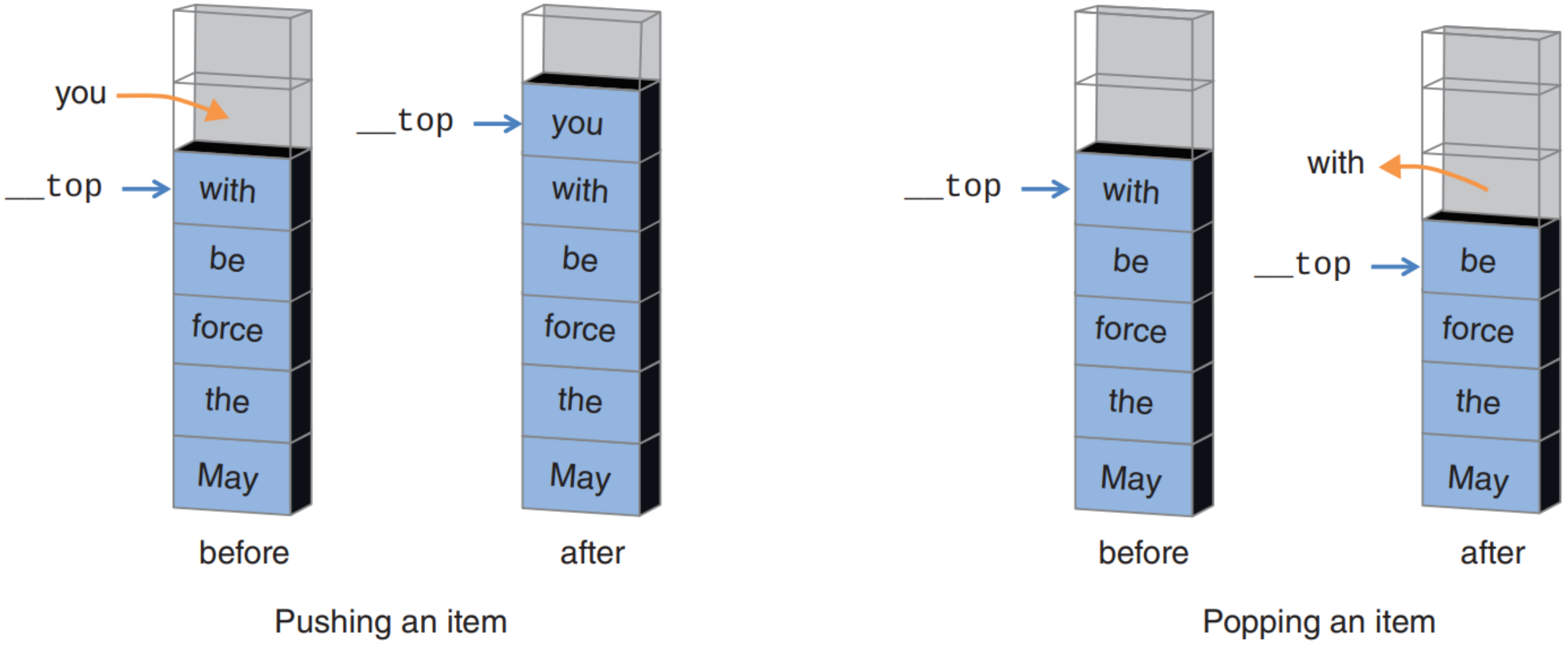
```python
for word in ['May', 'the', 'Force', 'be', 'with', 'you']:
    #s.push(word)
    if not s.isFull():
        s.push(word)
    else:
        print("Can't insert, stack is full")
```

STACK APPLICATION

1. Reverse a Word
2. Decimal To Binary
3. Parentheses Check

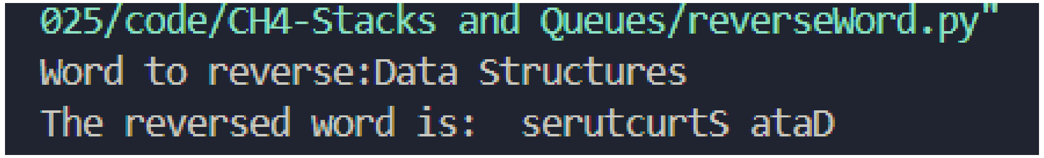
Example 1: Reverse a Word

- Take advantage of stack’s reverse property

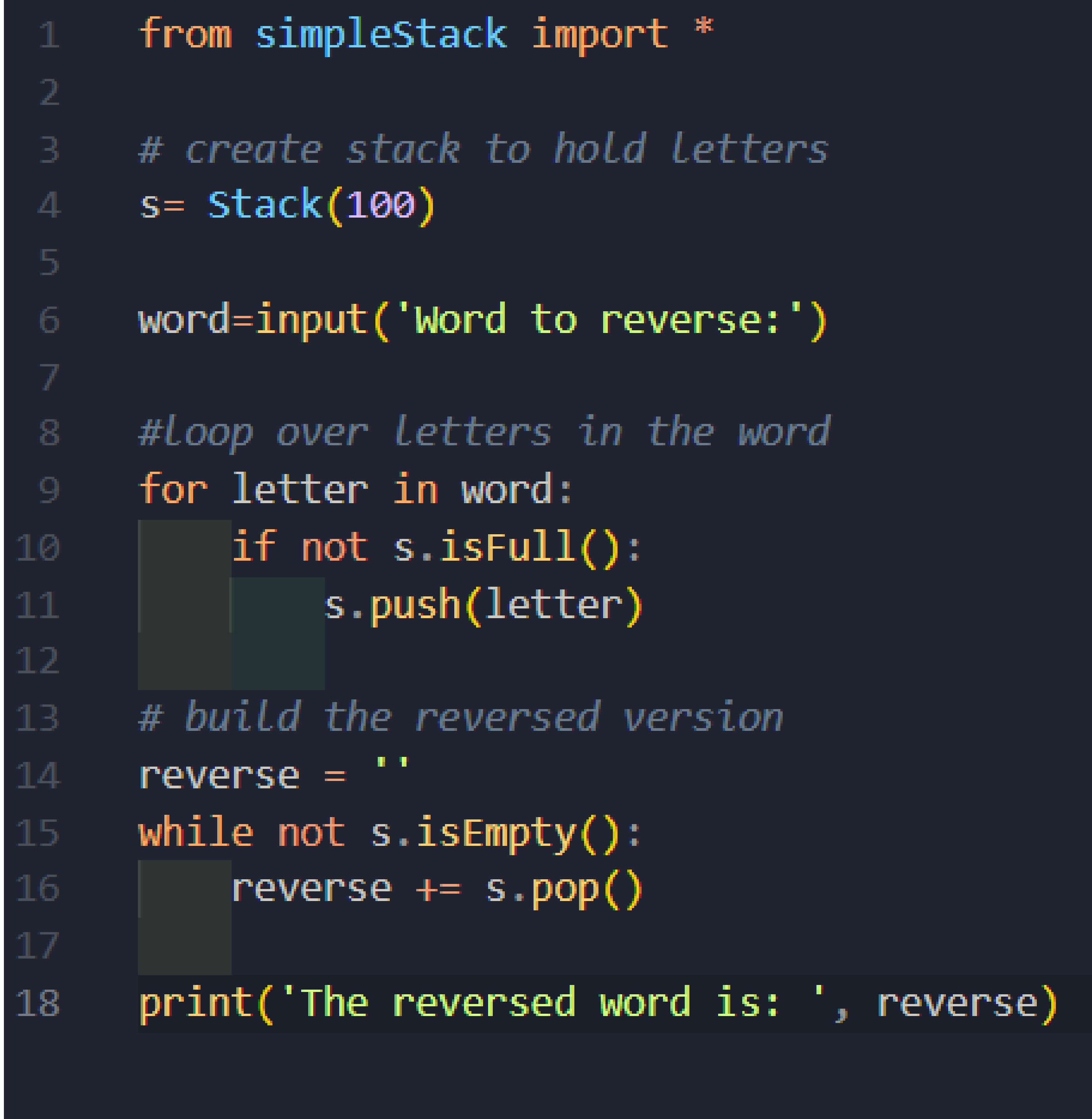


REVERSE A WORD

A stack is employed to invert the order of the letters.



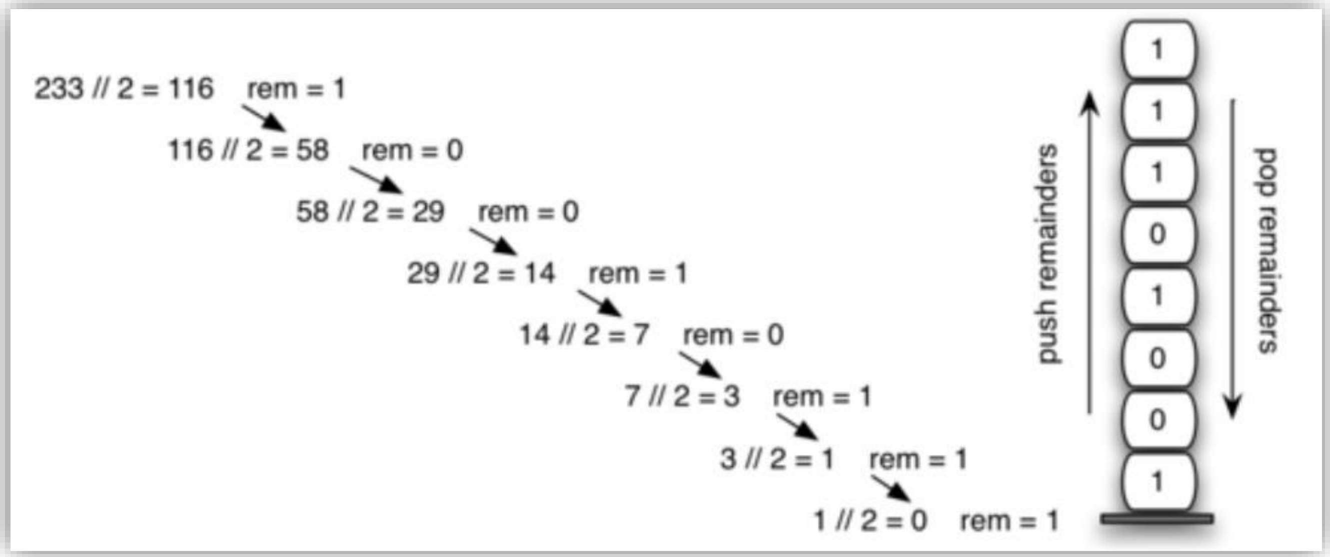
```
025/code/CH4-Stacks and Queues/reverseWord.py"
Word to reverse:Data Structures
The reversed word is:  serutcurtS ataD
```



```python
from simpleStack import *

# create stack to hold letters
s= Stack(100)

word=input('Word to reverse:')

#loop over letters in the word
for letter in word:
    if not s.isFull():
        s.push(letter)

# build the reversed version
reverse = ''
while not s.isEmpty():
    reverse += s.pop()

print('The reversed word is: ', reverse)
```

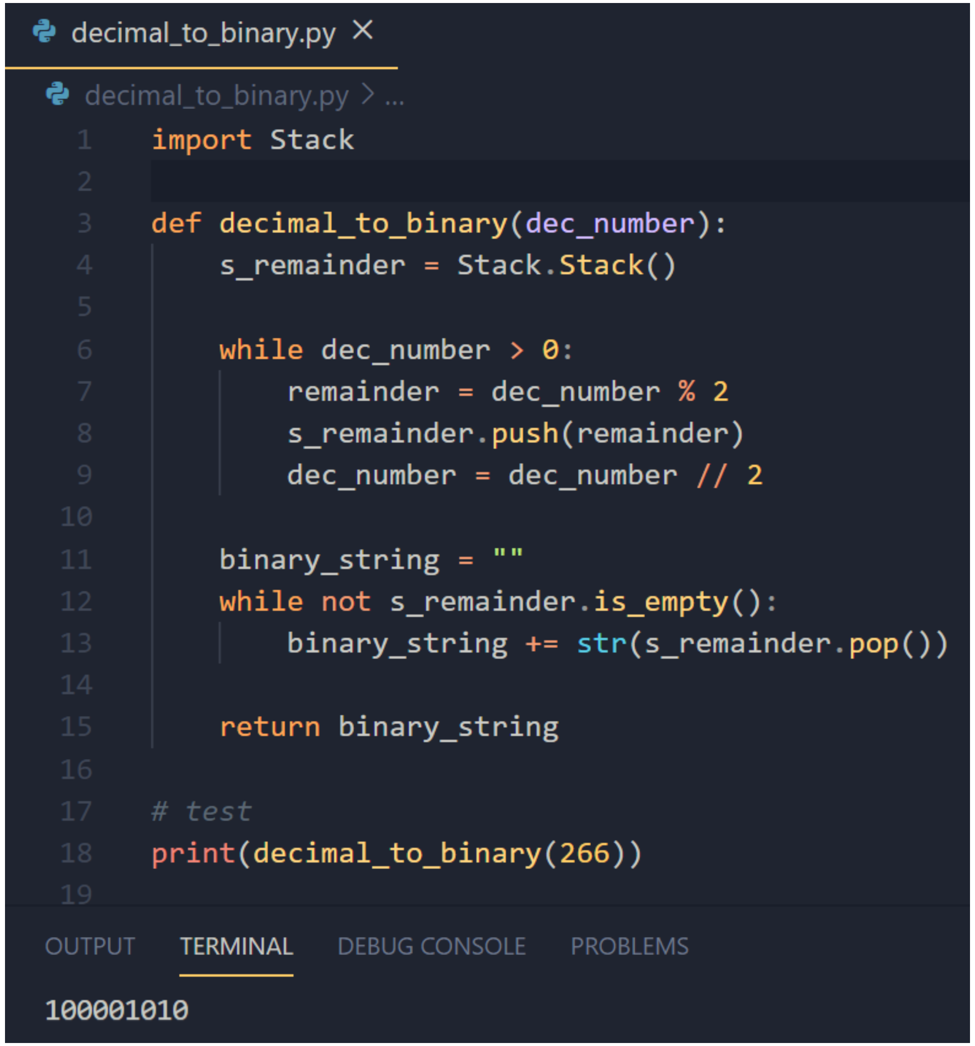
Example 2: Decimal To Binary

- How can we easily convert integer values into binary numbers? The answer is an algorithm called "Divide by 2,"



Example 2: Decimal To Binary

- We will use a stack to keep track of the digits for the binary result

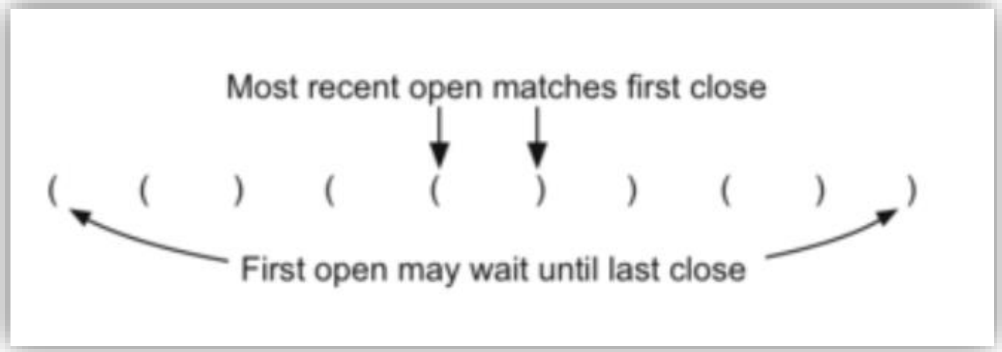


```python
import Stack

def decimal_to_binary(dec_number):
    s_remainder = Stack.Stack()

    while dec_number > 0:
        remainder = dec_number % 2
        s_remainder.push(remainder)
        dec_number = dec_number // 2

    binary_string = ""
    while not s_remainder.is_empty():
        binary_string += str(s_remainder.pop())

    return binary_string

# test
print(decimal_to_binary(266))

```

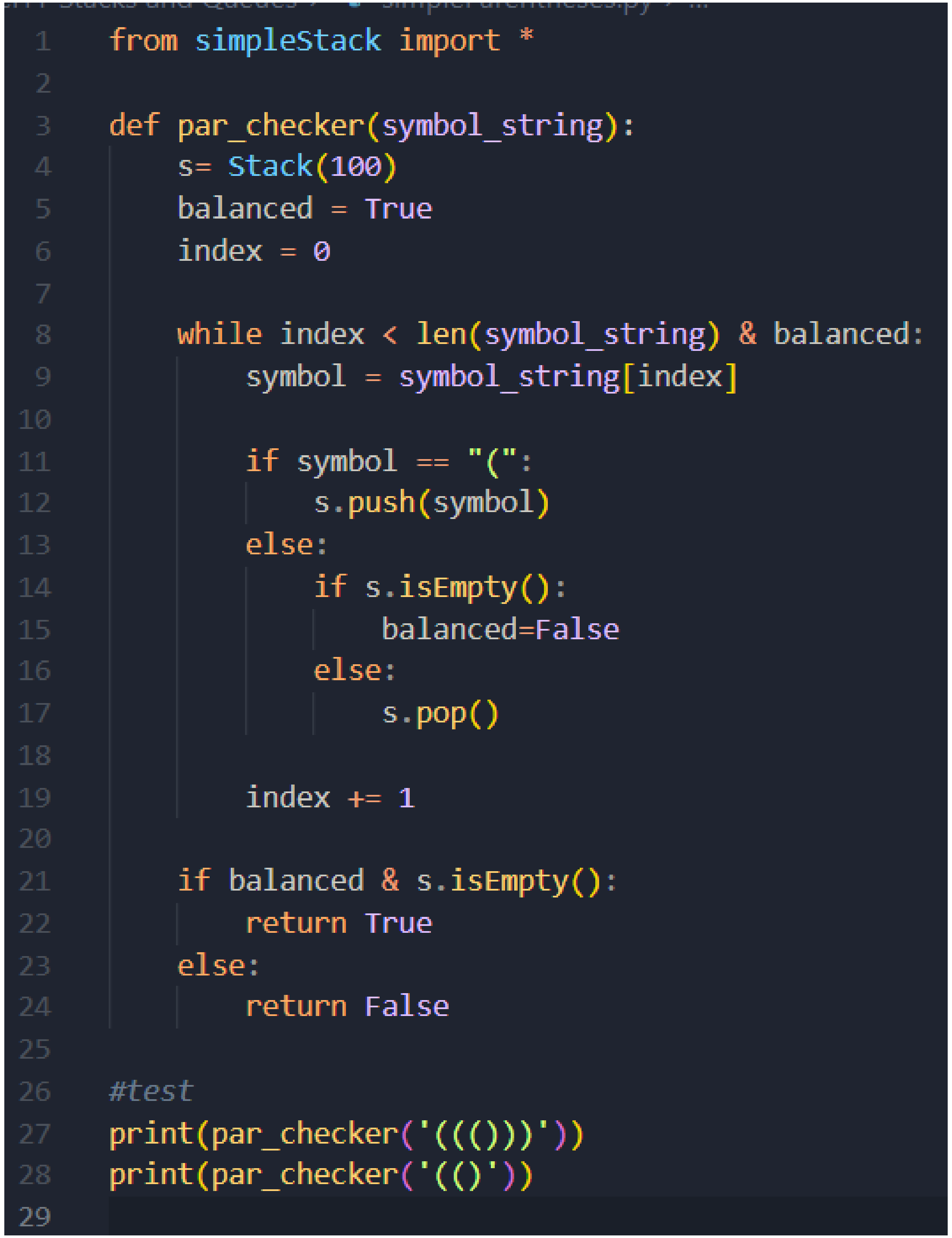
```
100001010
```

Example 3: Parentheses Checker

- . **Balanced parentheses** mean that each opening symbol has a corresponding closing symbol.
- (5 + 6)*(7 + 8)/(4 + 3)


Most recent open matches first close
(( ) (( )) ( ))
First open may wait until last close

EXAMPLE 3: PARENTHESES CHECKER



```python
from simpleStack import *

def par_checker(symbol_string):
    s= Stack(100)
    balanced = True
    index = 0

    while index < len(symbol_string) & balanced:
        symbol = symbol_string[index]

        if symbol == "(":
            s.push(symbol)
        else:
            if s.isEmpty():
                balanced=False
            else:
                s.pop()

        index += 1

    if balanced & s.isEmpty():
        return True
    else:
        return False

#test
print(par_checker('((()))'))
print(par_checker('(()'))
```

THANKS