# Stack

All Versions

## Examples

### Creating a Stack class with a List Object

Using a list object you can create a fully functional generic Stack with helper methods such as peeking and checking if the stack is Empty. Check out the official python docs for using list as Stack here .

```
#define a stack class
class Stack:
    def __init__(self):
        self.items = []

    #method to check the stack is empty or not
    def isEmpty(self):
        return self.items == []

    #method for pushing an item
    def push(self, item):
        self.items.append(item)

    #method for popping an item
    def pop(self):
        return self.items.pop()

    #check what item is on top of the stack without removing it
    def peek(self):
        return self.items[-1]

    #method to get the size
    def size(self):
        return len(self.items)

    #to view the entire stack
    def fullStack(self):
        return self.items
```

An example run:

```
stack = Stack()
print('Current stack:', stack.fullStack())
print('Stack empty?:', stack.isEmpty())
print('Pushing integer 1')
stack.push(1)
print('Pushing string "Told you, I am generic stack!"')
stack.push('Told you, I am generic stack!')
print('Pushing integer 3')
stack.push(3)
print('Current stack:', stack.fullStack())
print('Popped item:', stack.pop())
print('Current stack:', stack.fullStack())
print('Stack empty?:', stack.isEmpty())
```

Output:

```
Current stack: []
Stack empty?: True
Pushing integer 1
Pushing string "Told you, I am generic stack!"
Pushing integer 3
Current stack: [1, 'Told you, I am generic stack!', 3]
Popped item: 3
Current stack: [1, 'Told you, I am generic stack!']
Stack empty?: False
```

Improvements requested:

### Parsing Parentheses

Stacks are often used for parsing. A simple parsing task is to check whether a string of parentheses are matching.

For example, the string ([]) is matching, because the outer and inner brackets form pairs. ()<>) is not matching, because the last ) has no partner. ([)] is also not matching, because pairs must be either entirely inside or outside other pairs.

```
def checkParenth(str):
    stack = Stack()
    pushChars, popChars = "<({[", ">)}]"
    for c in str:
        if c in pushChars:
            stack.push(c)
        elif c in popChars:
            if stack.isEmpty():
                return False
            else:
                stackTop = stack.pop()
                # Checks to see whether the opening bracket matches the closing one
                balancingBracket = pushChars[popChars.index(c)]
                if stackTop != balancingBracket:
                    return False
        else:
            return False

    return not stack.isEmpty()
```

## Syntax

```
stack = [] # Create the stack
```

```
stack.append(object) # Add object to the top of the stack
```

```
stack.pop() -> object # Return the top most object from the stack and also remove it
```

```
list[-1] -> object # Peek the top most object without removing it
```

## Parameters

## Remarks

From Wikipedia :

> In computer science, a *stack* is an abstract data type that serves as a collection of elements, with two principal operations: *push* , which adds an element to the collection, and *pop* , which removes the most recently added element that was not yet removed.

Due to the way their elements are accessed, stacks are also known as *Last-In, First-Out* ( *LIFO* ) *stacks* .

In Python one can use lists as stacks with `append()` as push and `pop()` as pop operations. Both operations run in constant time O(1).

The Python's `deque` data structure can also be used as a stack. Compared to lists, `deque` s allow push and pop operations with constant time complexity from both ends.