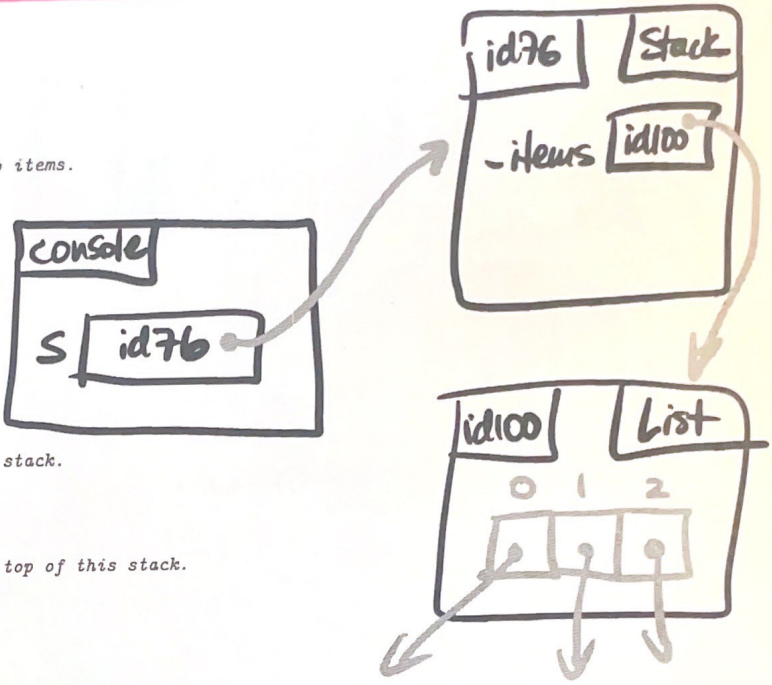
WK4 2411131 Chengmiao Yu

# CSC148 - A client function for class Stack: size

```
class Stack:
    """A last-in-first-out (LIFO) stack of items.

    Stores data in a last-in, first-out order. When removing an item from the
    stack, the most recently-added item is the one that is removed.
    """
    # === Private Attributes ===
    # _items:
    #     The items stored in this stack. The end of the list represents the top of the stack.
    _items: List

    def __init__(self) -> None:
        """Initialize a new empty stack."""

    def is_empty(self) -> bool:
        """Return whether this stack contains no items.

        >>> s = Stack()
        >>> s.is_empty()
        True
        >>> s.push('hello')
        >>> s.is_empty()
        False
        """

    def push(self, item: Any) -> None:
        """Add a new element to the top of this stack.
        """

    def pop(self) -> Any:
        """Remove and return the element at the top of this stack.

        >>> s = Stack()
        >>> s.push('hello')
        >>> s.push('goodbye')
        >>> s.pop()
        'goodbye'
        """
```




We are writing client code and need a function (outside the class) to determine the number of items on a stack.

1. Is the following a good solution? Explain.

```
def size(s: Stack) -> int:
    """Return the number of items in s.

    >>> s = Stack()
    >>> size(s)
    0
    >>> s.push('hi')
    >>> s.push('more')
    >>> s.push('stuff')
    >>> size(s)
    3
    """
    count = 0
    for _ in s:
        count += 1
    return count
```

a function (outside the class, no "self" parameter)

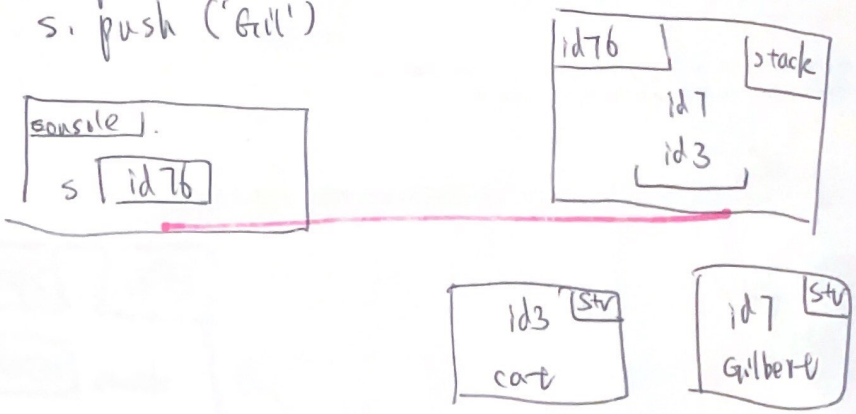
Class we define, are not automatically "iterable". → but it can be done.

∴ error.

code 不对

My ans: It is not a good sol, since _items meant to be private. SO, we should have a method inside the class

```
s = Stack()
s.push('cat')
s.push('Gill')
```



2. Is the following a good solution? Explain.

```
def size(s: Stack) -> int:
    """Return the number of items in s.
    """
    count = 0
    while not s.is_empty():
        s.pop()
        count += 1
    return count
```

I think it is correct, it uses 2 public methods.

does repeat the correct size BUT destroys the stack.

Bad. Don't do more than docstring says even if we think it's good.

3. Is the following a good solution? Explain.

```
def size(s: Stack) -> int:
    """Return the number of items in s.
    """
    return len(s._items)
```

ANS: Yes, but we should not use public attribute.

gives correct answer. BUT accesses a private instance variable

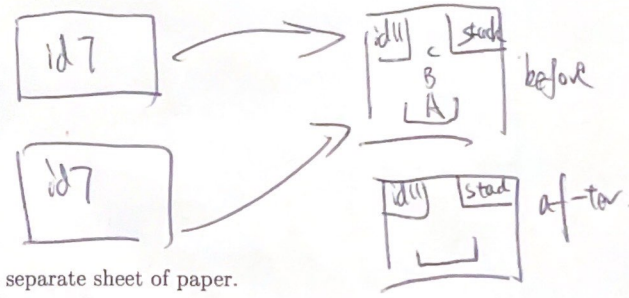
∴ function vulnerable.

4. Is the following a good solution? Explain.

```
def size(s: Stack) -> int:
    """Return the number of items in s.
    """
    s_copy = s
    count = 0
    while not s_copy.is_empty():
        s_copy.pop()
        count += 1
    return count
```

answer correct, but still empty the stack.

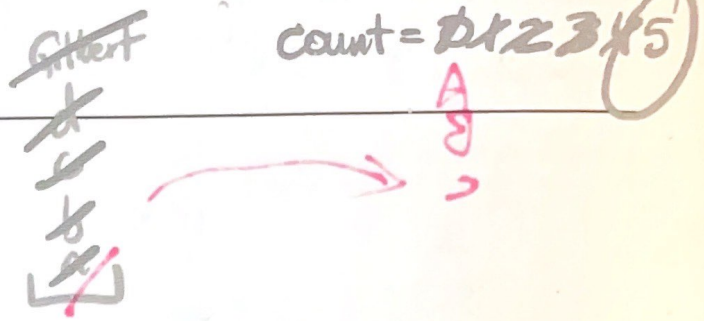
Does give correct answer.




5. Given what you've learned, implement the function yourself on a separate sheet of paper.

2. Is the following a good solution? Explain.

```
def size(s: Stack) -> int:
    """Return the number of items in s.
    """
    count = 0
    while not s.is_empty():
        s.pop()
        count += 1
    return count
```




does repeat the correct size

BUT destroys the stack.

Bad. Don't do more than docstring says even if we think it's good.

3. Is the following a good solution? Explain.

```
def size(s: Stack) -> int:
    """Return the number of items in s.
    """
    return len(s._items)
```

ans: Yes, but we should not use public attribute.

gives correct answer. BUT accesses a private instance variable

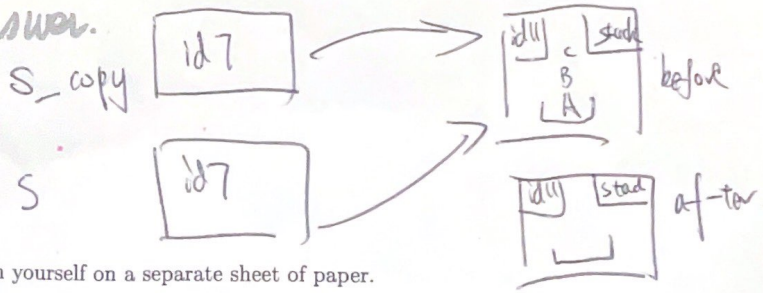
∴ function vulnerable.

4. Is the following a good solution? Explain.

```
def size(s: Stack) -> int:
    """Return the number of items in s.
    """
    s_copy = s
    count = 0
    while not s_copy.is_empty():
        s_copy.pop()
        count += 1
    return count
```

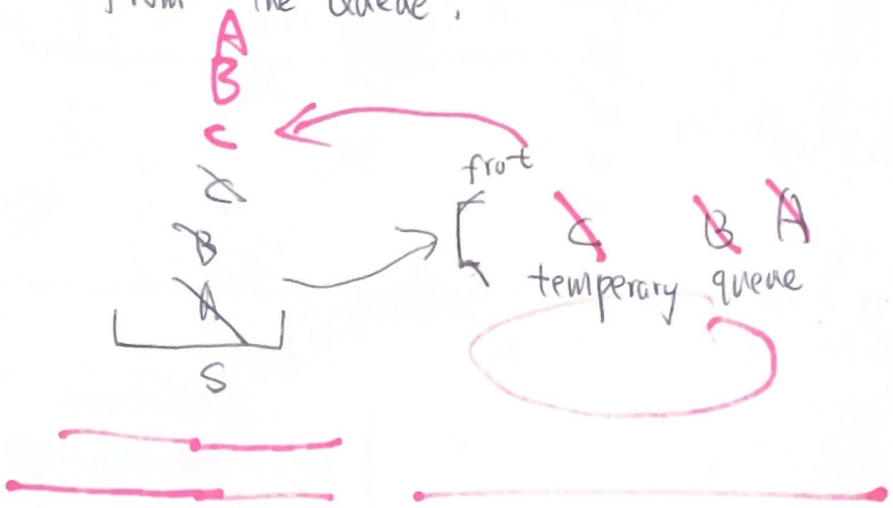
answer correct, but still empty the stack.

Does give correct answer.




5. Given what you've learned, implement the function yourself on a separate sheet of paper.

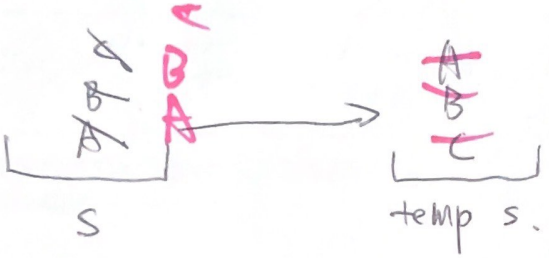
Can we solve this by popping into a Queue + restore from the Queue?




顺序不对

what a temp stack?

It works



WK4

# CSC148 - Balancing Parentheses

We are writing client code and need a function (outside the class) to determine whether the parentheses in an expression are balanced: opening and closing parentheses match and are properly nested inside each other.

1. For four examples, we'll give you a string one character at a time. Your job is to determine whether the string has balanced parentheses or not. *Don't just write down every character without thinking!* Instead, use a stack to keep track of the minimum amount of information you need to solve the problem.

Expression 1:

Balanced ( ) | ( ( 2 ( 3 4 ) ) | ( ( ( 1 1 ) ) | ( ) ( )

not ( ) ( ( ) | ( ( | ( ) ( ) )

Stack

Were the parentheses balanced?

Yes No

Expression 2:

) ) ) ) ( ( ( (

Stack

Were the parentheses balanced?

Yes No

Expression 3:

( )

Stack

Were the parentheses balanced?

Yes No

Expression 4:

Stack

Were the parentheses balanced?

Yes No

2. We need a general strategy that will work in all cases. To find it, answer these questions:

   (a) What will you do with each character as you receive it?

   ( : push

   ) : pop

   anything else: ignore

   (b) At the end, how will you know whether the parentheses were balanced?

   See "rough work" for more

3. Now implement the function.

```
def is_balanced(line: str) -> bool:
    """Return whether <line> contains balanced parentheses.

    Ignore square and curly brackets.

    >>> is_balanced('(a * (3 + b))')
    True
    >>> is_balanced('(a * (3 + b]]')  # Note that the two ']'s don't matter.
    False
    >>> is_balanced('1 + 2(x - y)}')  # Note that the '}' doesn't matter.
    True
    >>> is_balanced('3 - (x')
    False
    """
```

① )6(

what if we pop if stack is empty, pop:

if stack is not empty → pop

is empty → return False

((3(

↑ not empty at end

(1(1 ↑ empty at end

return is_empty

② keep one int,

( then +1

) then −1.

int < 0 return F

```
i = 0
unmatched = Stack()
for char in line:
    if c == '(':
        unmatch.push('(')
    elif c == ')':
        if unmatch.is_empty():
            return False
        else:
            unmatch.pop()
return unmatch.is_empty()
```

```
i = 0
tracker = 0.
for char in line:
    if tracker < 0:
        return False.
    if char == '(':
        traker += 1
    if char == ')':
        track -= 1
return tracker == 0.
```

4. How would you generalize this code to balance round, square, and curly brackets?

.pop() cause error when pop from empty stack.

```
class PopEmptyStackError(Exception):
    pass
    def __str__(self) -> str:
        return 'Never pop emty stack'
```

~~if self.~~

```
def pop(self) -> Any:
    if self.is_empty():
        raise PopEmpyStackError.
```

2/2 Albert <hongmiao Yu

# CSC148 - Considering a different implementation of class Stack

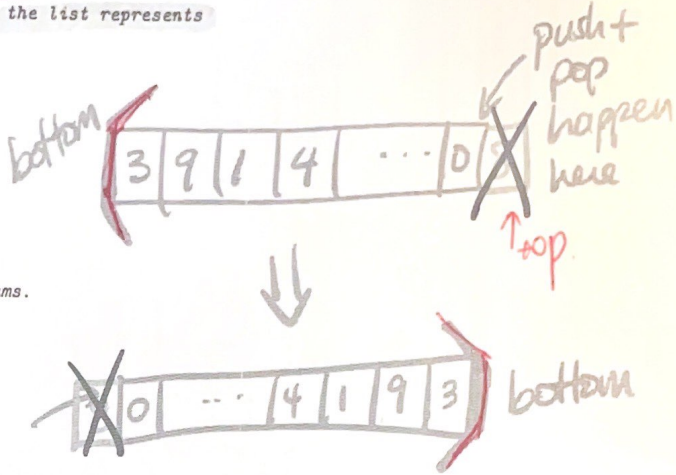
Below is the complete stack class that you saw in the readings.

Stack2.

```
class Stack:
    """A last-in-first-out (LIFO) stack of items.

    Stores data in first-in, last-out order. When removing an item from the
    stack, the most recently-added item is the one that is removed.
    """
    # === Private Attributes ===
    # _items:
    #     The items stored in the stack. The end of the list represents
    #     the top of the stack.
    _items: List

    def __init__(self) -> None:
        """Initialize a new empty stack.
        """
        self._items = []

    def is_empty(self) -> bool:
        """Return whether this stack contains no items.

        >>> s = Stack()
        >>> s.is_empty()
        True
        >>> s.push('hello')
        >>> s.is_empty()
        False
        """
        return self._items == []

    def push(self, item: Any) -> None:
        """Add a new element to the top of this stack.
        """
        self._items.append(item)

    def pop(self) -> Any:
        """Remove and return the element at the top of this stack.

        >>> s = Stack()
        >>> s.push('hello')
        >>> s.push('goodbye')
        >>> s.pop()
        'goodbye'
        """
        return self._items.pop()
```

Don't forget to check the R.I's.

front




push+pop here

To make the front of list represent top insert the item to list at index 0.

insert(0, item)

pop(0)

0

```
def f1() -> None:
    f2()

def f2() -> None:
    f3()

def f3() -> None:
    x = input('Num: ')
    print(100 / int(x))
```

更新后的 f(3)

```
def f(3)
    try:
        ...
    except ZeroDivisionError:
        print
    except ValueError:
        print. 等输入是int.
```

ls = [3, 8, 'A', 2]

想 catch error:

```
try:
    total = 0
    for it in ls:
        total += it
except ValueError:
    pass
return total
```

res: 59.

WKS

1. We are going to revise the code so that instead of putting the top of the stack at the end of the list, the top of the stack is at the *beginning* of the list.

   (a) What code has to change?

done

   (b) Make the changes. done

2. What docstrings must change to go along with your code changes? Explain.

done

3. What changes would be required to the parenthesis balancer code that we worked on in the last lecture in order for it to work with this new version of the class? Explain.

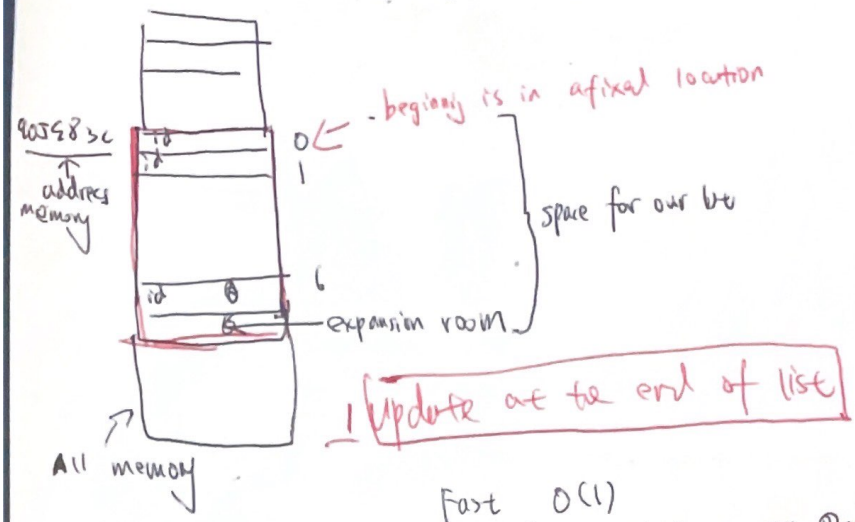
None run it + see.

4. Which implementation do you think is better? What criteria are you using?

faster is better. Is one version of stack faster ??
≡ stability (easy to change without breaking)
less memory is better ≡
≡ less error prone → clean, ≡ elegant, simple

How a python list is stored.

list of 7: [3, 9, 22, 0, 6, 0, 14] 创建 list 时，内存会给一些空余空间




1 Update at the end of list

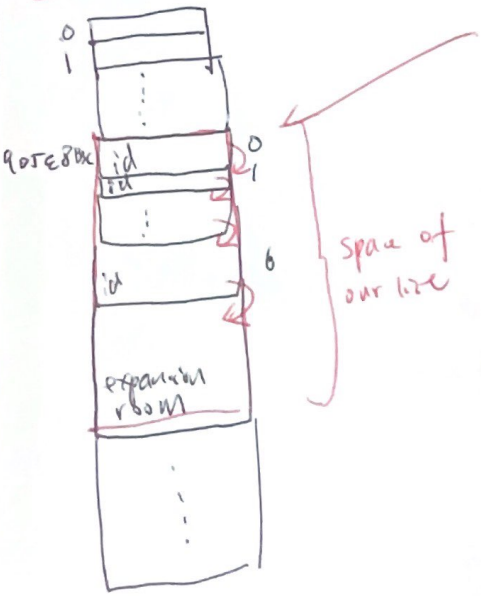
Fast O(1)

Index into our list：只是算术，因为知道 list 开头的 memory address 905e83c，要去 list 的 index k，只需去 memory add 905e83c + k 就可。

Append to our list：也 Fast. 可以用 expansion room，until run out of expansion room.

Delete at the bottom：Fast. 去掉 last one

2 update at the frone of the list.




① 在 front of list 加 item，需从倒最后一个 index 开始直到 index 0，依序向后挪 1 个 position，最后在 index 0 腾出一个空位

very slow.