# 清风塾 CSC148 Term Test 1 Review

2017年2月4日 0:55

1. 关于Test 1
   - Recursion：考output，并且一定是和**list comps**合用的一个**tracing question**
   - Abstract data types：只学了stack，queue和sack
   - Class design：关于inheritance的用法，superclass和subclass中方法的优先级，override的注意事项。单独拿出来考的可能性不大，通常情况下，应该是题目中的一部分。

   * Exception???

2. 列表解析 List Comprehensions / List Comps

   **语法：**
   ```
   [expr for iter_var in iterable]
   ```

   **语法：**
   ```
   [expr for iter_var in iterable if cond_expr]
   ```

   **语法：**
   ```
   [expr if cond_expr else expr for iter_var in iterable]
   ```

3. 生成器表达式

   **语法：**
   ```
   (expr for iter_var in iterable)
   ```

   **语法：**
   ```
   (expr for iter_var in iterable if cond_expr)
   ```

   **语法：**
   ```
   (expr if cond_expr else expr for iter_var in iterable)
   ```

4. all()：全部为True才是True

   Return True if all elements of the iterable are True (or if the iterable is empty).

   any()：一个是True就是True

   Return True if any element of the iterable is True. If the iterable is empty, return False

5. 关于if, else的禅用法 **要求掌握！！！！**

   **语法：**
   ```
   if expr: expr_true_statement
   ```

   **语法：**
   ```
   expr_true_statement if expr else expr_false_statement
   ```

6. 练习：

   6.0 Given a list of n integers called lst. Return the result of the difference of the sum of the numbers on odd indices and even indices.（求奇数位数数字之和与偶数位数字之和的差）

   **答案：**

   6.1 Given a matrix with dimension m * n, return a list of tuples of all coordinates(starts with(0, 0))

```
     [,1] [,2] [,3] [,4]
[1,]  Inf  Inf  Inf  Inf
[2,]  Inf  Inf    0  Inf
[3,]  Inf  Inf  Inf  Inf
[4,]  Inf  Inf  Inf  Inf
[5,]  Inf  Inf  Inf  Inf
```

**答案：**

6.2 Given a list contains a number of strings call lst, return True iff all strings on odd indices are "Yes" and all strings on

even indices are "No". Example:
["Yes", "No", "Yes", "No", "Yes", "No", "Yes"] will return True

**答案：**

## 7. Recursion递归

**如果函数包含了对其自身的调用，该函数就是递归的。**A function is recursive if it contains a call to itself。（间接递归我们148不讨论），我们的目的是为了把问题分成小份儿，然后调用函数本身来解决这些分解以后的小份儿。

注意笔记中有一句特别重要，对于较大或较为复杂的问题，往往递归前后需要进行一些处理。

`In some cases, we need to combine the solution!`

### Recursion types 递归分类

- Depends on how we split the problem
  - "N-1" approach: handle one entity, then call the recursion for N-1 entities 将整体切分出一小部分（一般是整体的单个最小单位，比如list的一个元素），然后对整体剩余的部分做递归。
  - Divide in 2 or more subproblems: apply recursion for each half, quarter, etc. of the problem 将问题均分为两份或更多等份，然后再均分后的没小份上进行递归。注意，很多时候这个过程，会不停地拆分，直至拆分成最小单位（如N-1中提到的）
  - Other ways (more later..) 暂时不做讨论
- Recursion has 2 phases/steps:
  1. Base case (这是最容易完成的部分)
     - Simplest problem, cannot break it down further
     - This is where we stop recursing
  2. Recursive decomposition step
     - Breaks down the problem into smaller, "similarly-solvable" subproblems
     - Must guarantee to eventually get to the base case (要能自缩小，直到base case)

Iteration 和 Recursion 都是一种循环，iteration实际上是在固定次数内循环。而recursion是在函数本身循环，从输入的情况开始，经过recursive step不断缩小，并进入更深一层，直至触及base case，然后将值逐步返回上层得出结果。

**特别重要的一点：你在做recursion之前，一定要非常清楚你在做的这个function是做什么的！！其次，能将问题抽象，将方法概括成一句精简的描述。比如Depth of a list = 1 plus the maximum depth of L’ s elements if L is a list, otherwise its depth is 0.**

**先做一道比较复杂的，要求写function的题目**

Question 2. [15 marks]

Python's `str.join` concatenates a list of strings, using the given separator string, and returns the new

string, as follows:

```
>>> str.join('+',['one', 'two', 'three'])
    'one+two+three'
```

Complete the denition of `nested_join(...)`, which concatenates the strings in a nested list of strings, using the given separator string, and returns the result.

```
def nested_join(s: str, L: list) -> str:
    """Return join of nested list of strings L with separator string
    s
    >>> nested_join(' ', [])
    ''
    >>> nested_join(' ', ['one'])
    'one'
    >>> nested_join(' ', ['one', 'two'])
    'one two'
    >>> nested_join(' ', ['one', ['two', 'three'], 'four'])
    'one two three four'
    """
solution:
```

```
def sum_list(L):
    if len(L) == 0:
        return 0
    else:
        return L[0] + sum_list (L[1:])
```

- L = [1, [5,3], 8, [4,[9,7]]]

```
def sum_list(L):
    if isinstance(L, list):          # recursive step
        return sum([sum_list(elem) for elem in L])
    else:                            # base case
        return L
```

- Example: L = [1, [5,3], 8, [4,[9,7]]]

```
def depth(L):
    if not isinstance(L, list):      # base case
        return 0
    else:                            # recursive step
        return 1 + max([depth(elem) for elem in L])
```

Question 1. [5 marks]

Read over the denition of this Python function:

```
def c(s):
    """Docstring (almost) omitted."""
    return sum([c(i) for i in s]) if isinstance(s, list) else 1
```

我们先翻译这个function

```
    if isinstance(s, list):
        return sum([c(i) for i in s])
    else:
        return 1
"""Count the TOTAL number of elements in the nested list"""
```

Work out what each function call produces, and write it in the space provided.

做第一题有两种办法，一种是真的代数，一种是理解题意。代数容易来不及写完后边的题目，自己看着办。

1. c(5)
1
2. c([])
0
3. c(["one", 2, 3.5])
3
4. c(["one", [2, "three"], 4, [5, "six"]])
6
5. c(["one", [2, "three"], 4, [5, [5.5, 42], "six"]])
8

---

Question 1. [5 marks]
Read over the denition of this Python function:

```
def c(n):
    """Docstring (almost) omitted."""
    return sum([c(i) for i in n]) if isinstance(n, list) else n
```

**"""Get the sum of  all elements in the nested list"""**

Work out what each function call produces, and write it in the space provided.

1. c(5)

2. c([])

3. c([1, 2, 3.5])

4. c([1, [2, 3], 4, [5, 6]])

5. c([1, [2, 3], 4, [5, [5.5, 42], 6]])

---

Question 1. [5 marks]
Read over the denition of this Python function:

```
def c(n):
    """Docstring (almost) omitted."""
    return 1 + max([c(i) for i in n] + [0]) if isinstance(n, list) else 0
```

**"""Get the depth of the nested list"""**

Work out what each function call produces, and write it in the space provided.

1. c(5)

2. c([])

3. c([1, 3, 5])

4. c([0, [1, 3, 5], 7])

5. c([0, [1, 3, 5, [7, [9]]], 11])

---

Question 1. [5 marks]
Read over the denition of this Python function:

```
def c(n):
    """Docstring (almost) omitted."""
    return max([len(n)] + [c(i) for i in n]) if isinstance(n, list) else 0
```

**"""Get the length of longest elements in the nested list"""**

Work out what each function call produces, and write it in the space provided.

1. c(5)

2. c([])

3. c([1, 3, 5])

4. c([0, [1, 3, 5], 7])

5. c([0, [1, 3, 5, [7, [9]]], 11])

---

Question 1. [12 marks]
I have left out the documentation for function s(n). Work out what it produces, starting at the smallest n and working up.

```
def s(n: int) -> tuple:
    """For you to figure out..."""
    if n == 1:
        return (1, 1)
    else:
        return min([(2 * s(n - i)[0] + 2**i - 1, i) for i in range(1,n)])
```

Part (a) [2 marks]
s(1):
Part (b) [2 marks]
s(2): (3,1)
Part (c) [2 marks]
s(3): (5,2)
Part (d) [2 marks]
s(4): (9,2)
Part (e) [2 marks]
s(5): (13,2)
Part (f) [2 marks]
s(6): (17,3)

8. **Abstract Data Type**

**考试应该会围绕已经学过的Stack, Queue和Sack来出题，implementation是一个有很大机会会出现的题型。**

## Question 4. [5 MARKS]

Complete the implementation of **push** in the class **DividingStack**, a subclass of **Stack**. Notice that you may use **push**, **pop**, and **is_empty**, the public operations of **Stack**, but you may not assume anything about **Stack**'s underlying implementation. You may find it useful to know that if **n1** and **n2** are integers, then **n1** % **n2** == 0 if and only if **n2** divides **n1** evenly.

```
from csc148stack import Stack
"""
Stack operations:
    pop(): remove and return top item
    push(item): store item on top of stack
    is_empty(): return whether stack is empty.
"""

class DividingStack(Stack):
    """A stack of integers that divide predecessors."""

    def push(self: 'DividingStack', n: int) -> None:
        """Place n on top of self provided it evenly divides its predecessor.
        Otherwise, raise an Exception and leave self as it was before


        precondition - possibly empty self contains only integers

        >>> s = DividingStack()
        >>> s. push(12)
        >>> s.push(4)
        >>> # now s.push(3) should raise Exception
        """
```

## Question 4. [5 MARKS]

Complete the implementation of **push** in the class **PrefixStack**, a subclass of **Stack**. Notice that you may use **push**, **pop**, and **is_empty**, the public operations of **Stack**, but you may not assume anything about **Stack**'s underlying implementation. You may find it useful to know that if **s1** and **s2** are strings, then **s1.startswith(s2)** returns True if **s2** is a prefix of **s1**, and False otherwise.

```
from csc148stack import Stack
"""
Stack operations:
    pop(): remove and return top item
    push(item): store item on top of stack
    is_empty(): return whether stack is empty.
"""


class PrefixStack(Stack):


    """Stack of strings where each is a prefix of its predecessor"""

    def push(self: 'PrefixStack', s: str) -> None:
        """Place s on top of stack self, provided s is a prefix of
        its predecessor.  Otherwise raise an Exception and leave
        stack self as it was

        precondition - possibly empty self contains only strings

        >>> s = PrefixStack()
        >>> s.push("asterisk")
        >>> s.push("aster")
        >>> # now s.push("asteri") should raise Exception
        """
```

---

Question 3. [15 marks]
You are to implement FunctionalList, a subclass of built-in class list.
FunctionalList should have
two new methods:

```
functional_append(self: 'FunctionalList', o: object) ->
'FunctionalList':
    """Return a copy of this FunctionalList with o appended"""
functional_sort(self: 'FunctionalList') -> 'FunctionalList':
    """Return a sorted copy of this FunctionalList"""
```

Note that functional_append and functional_sort are not allowed to
change the original list they are
called on.

Part (a) [8 marks]
Implement FunctionalList in the space below.

**solution:**

```
    """list with some functional methods."""
    def functional_append(self: 'FunctionalList', o: object) ->
    'FunctionalList':
        """Return a copy of this list with o appended"""
        return FunctionalList([x for x in self] + [o])
    def functional_sort(self: 'FunctionalList') -> 'FunctionalList':
        """Return a sorted copy of this FunctionalList"""
```

```
Part (b) [7 marks]
Describe three more test cases that would increase your condence that
functional_append and functional_sort work as specified. One example is
given.
    Create FL = FunctionalList([1]). Then FL.functional append(2)
    returns the FunctionalList [1, 2] and FL is left unchanged.
```

9. Class Design

因为此处涉及到的内容篇幅太长，最好的办法是做2016Fall.Test1的真题。

10. Exception 错误和异常

这里定义忽略不写，大家看下笔记，今次我们主要讨论用法和真题

**语法：**

```
try:
    try_suite       #监控这里的异常
except Excetipion[, reason]:
    except_suite    #处理异常的代码
```

这个方式可以有效地捕捉异常，在所有except包含的异常发生时，程序不会因此而停止，而是会执行except_suite所对应的部分。

我们这里只做最简要的介绍，如果想进一步阅读可以看下我推荐的书，或者 https://docs.python.org/3/tutorial/errors.html

据说这个叫一锅出。。。这个模板展示了所有try-excpt-else-finally的用法和情况，可以参考一下。

**语法：**

```
try:
    try_suite
except Exception1:
    suite_for_Exception1
except (Exception2, Exception3, Exception4):
    suite_for_Exceptions_2_3_and_4
except Exception5, Argument5:
    suite_for_Exception5_plus_argument
except (Exception6, Exception7), Argument67:
    suite_for_Exceptions6_and_7_plus_argument
except:
    suite_for_all_other_exceptions
else:
    no_exceptions_detected_suite
finally:
    always_execute_suite
```

**raise 语句**

**关于raise的具体用法非常广泛，我们只考虑和if连用的情况。**

**语法：**

```
raise [SomeException [, args [, traceback]]]
```

**Question 2. [5 marks]**

**Read over the declarations of the three Exception classes, the denition of raiser, and the supplied code for notice below. Then complete the code for notice, using only except blocks, and perhaps an else block.**

```
class SpecialException(Exception):
    pass
class ExtraSpecialException(SpecialException):
    pass
class UltraSpecialException(ExtraSpecialException):
    pass
def raiser(s: str) -> None:
    """Raise exceptions based on length of s."""
    if len(s) < 2:
        raise SpecialException
    elif len(s) < 4:
        raise ExtraSpecialException
    elif len(s) < 6:
        raise UltraSpecialException
    else:
        b = 1 / int(s)

def notice(s: str) -> str:
    """Return messages appropriate to raiser(s).
    >>> notice("123456")
    'ok'
    >>> notice("000000")
    'exception'
    >>> notice ("12345")
    'ultraspecialexception'
    >>> notice("123")
    'extraspecialexception'
    >>> notice("1")
    'specialexception'
    """
```