Disclaimer

This complementary study package is provided by Easy 4.0 Education Inc. and its affiliated mentors. This study package seeks to support your study process and should be used as a complement, **NOT** substitute to course material, lecture notes, problem sets, past tests and other available resources.

We acknowledge that this package contains some materials provided by professors and staff of the University of Toronto, and the sources of these materials are cited in details wherever they appear.

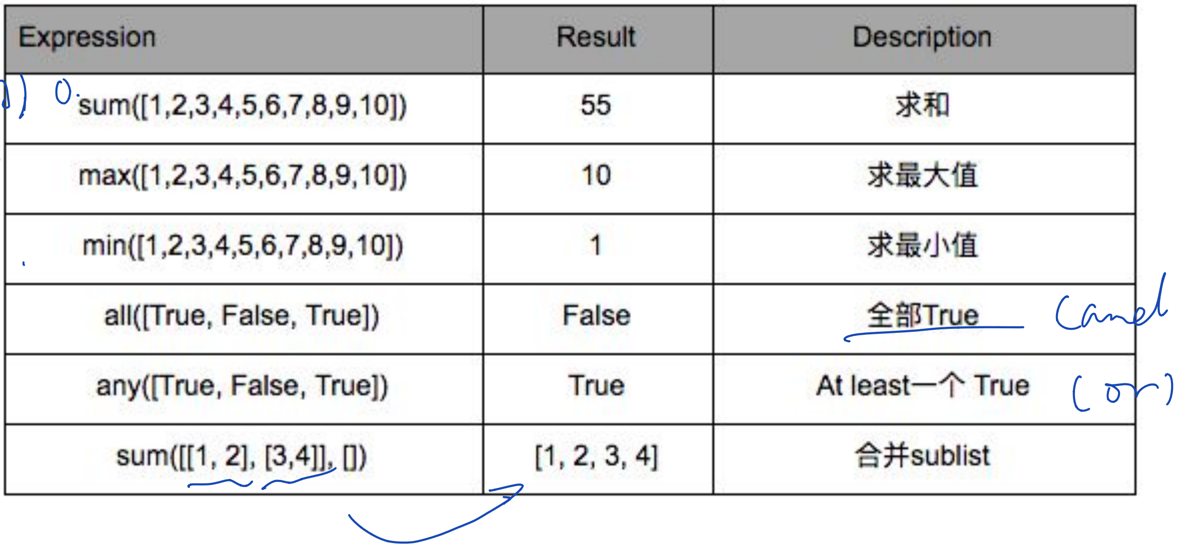
This package is distributed for free to students participating in Easy 4.0's review seminars, and are not for sale or other commercial uses whatsoever. We kindly ask you to refrain from copying or selling in part or in whole any information provided in this package.

Thank you for choosing Easy 4.0. We sincerely wish you the best of luck in all of your exams.

Easy 4.0 Education Inc.

Section 1. Coding Strategy

1. Build-in functions for iterables.



Expression	Result	Description
sum([1,2,3,4,5,6,7,8,9,10])	55	求和
max([1,2,3,4,5,6,7,8,9,10])	10	求最大值
min([1,2,3,4,5,6,7,8,9,10])	1	求最小值
all([True, False, True])	False	全部True
any([True, False, True])	True	At least一个 True
sum([[1, 2], [3,4]], [])	[1, 2, 3, 4]	合并sublist

Sum([]) 0.

不能是空的

(and)

(or).

Warning： Some of them do not work on some types of elements!
For example,You cannot get the sum of a list of string.

2. Type checking.

```
isinstance(x, A) :
   isinstance(object, class-or-type-or-tuple) -> bool

                    检查 x 是否是type A.
   Useful types: int, str, list, tuple, dict, etc……
   Example:
   >>> isinstance(123, int)
   True
```

2. List Comprehension.

Use one line of code to accomplish the construction of new list

G派心法: 表达式在先， 大家都来做， 要是有如果， 符合才通过， 后面接或者，
去做其他活儿，Loop接着放，定义好变量， 筛选很特殊， 条件最后讲！

a) No condition

[表达式 for loop]

b) Filter

[表达式 for loop if 条件]

c) If and else

[表达式 if 条件 else 其他活 for loop]

Example：

```
>>> lst = [1,2,3,4,5,6]
>>> [a for a in lst if a%2 == 0]   # get all even numbers
[2, 4, 6]
>>> lst = [1,2,3,4,5,6]
>>> [1 if a%2==0 else -1 for a in lst]
[-1, 1, -1, 1, -1, 1]
```

Question 1: use one line of code, get the sum of lst

```
>>> lst = [[1,2],[3,4],[5,6]]
>>> sum([sum(L) for L in lst]).
```

Question 2: use one line of code, get a list with all letters in uppercase

```
>>> lst = ['a', 'd', 'C', 'f', 'q', 'A']
>>> [c.upper() for c in lst]
```

Question 3: all string elements wanted!

```
>>> lst = ['e', 'z', 13, '4', 77, (66, 8)]
[item for item in lst if isinstance(item,
                                      str)].
```

Section 2. Recursion

Recursion是CSC148的核心内容，中文含义是递归，意指程序在运行当中调用自身，来完成一些复杂和庞大的计算模型。

Let’s compute $\sum_{n=1}^{50} n$

$$= 50 + \sum_{n=1}^{49} n.$$

$$= 50 + 49 + \cdots + \boxed{\sum_{n=1}^{1} n.} \quad 1.$$

$$\sum_{n=a}^{b} n = \begin{cases} a & a=b. \quad \text{\#Base Case,} \\ b + \boxed{\sum_{n=a}^{b-1}} n. & a<b \quad \text{\#Recursive} \end{cases}$$

Recursion 三大要素：

1. Function的作用：根据docstring 搞清function的具体作用, 在写body的时候assume function已经写好啦. input, output

2. Base Case：一般来说是各种数据类型的最小值,
   比如integer的一般为0或1.
   String的一般是empty string, nested list的一般是非list或者empty list.
   Tree: leaf   BT: None.

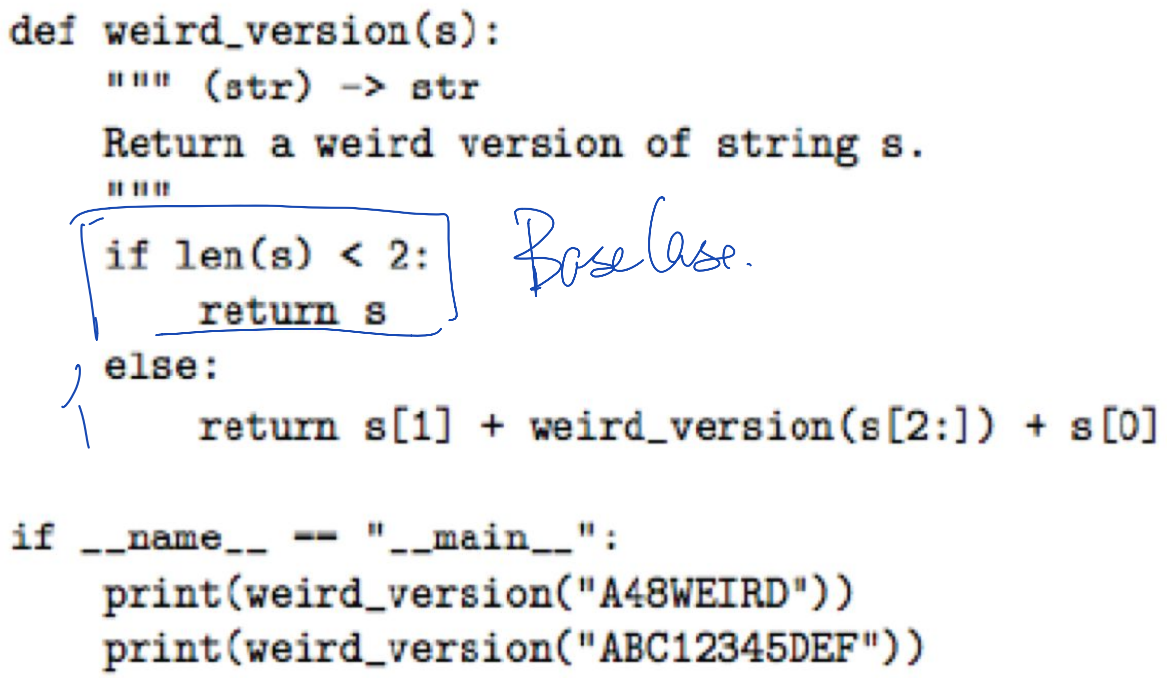
3. Recursive step:

3.1 Divide step： 搞清楚是如何把大问题化小，是如何把当前问题分解成一个相对较小的问题.   partial solution

3.2 Combination step： 搞清楚是如何把子问题的结果组合成原问题的结果.

G派心法: BaseCase先找到， 所有问题都需要，
recursive别多想， step by step找诀窍.

Trace recursion or Compute the result



```
def weird_version(s):
    """ (str) -> str
    Return a weird version of string s.
    """
    if len(s) < 2:
        return s
    else:
        return s[1] + weird_version(s[2:]) + s[0]

if __name__ == "__main__":
    print(weird_version("A48WEIRD"))
    print(weird_version("ABC12345DEF"))
```

Base Case.

4 + weird_version('8WEIRD') + A

4 W I D R, E 8 A

B 1 3 5 E F D 4 2 C A.

Part (b) [3 MARKS]

Suppose we made a new version of the function weird called weirdplus, where we changed the line
`if len(s) < 2:`
to
`if len(s) < 3:`
What can we say about a string s if we know that weird(s) != weirdplus(s)?

len(s) % 2 == 0

Nested List recursion

Nested list recursion是我们在148接触的第一种recursion的考点, 每一个nested list的recursion问题简单来说可以分为两种case:

1. 当前非list的情况.
2. 当前依然是list的情况.

[□, ☆, △]
0 0 0

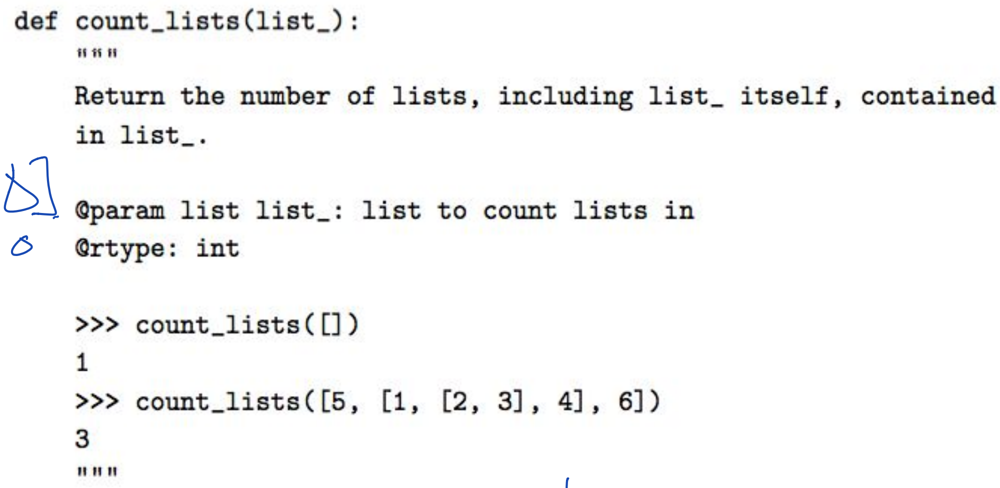
```
def count_items(lst):
    """Return the number of items in the nested list.
    """
    if not isinstance(lst, list):
            return 1.

    else:   acc= 0
            for sublist in lst:
                acc += count_items(sublist)
            return acc.

def flatten_items(lst):
    """Return the number of items in the nested list.
    """
    if not isinstance(lst, list):
        return [lst]
    else:
        acc= []
        for s in lst:
            acc += flatten_items(s)
    return acc
```

在recursion问题中, accumulation (combination)环节会根据题上不同要求存在不同的方法，例如求depth, 每次是取sub problem 的max 加1, 求总和是直接求和或者求 sum etc

[□, ☆, △]
8 7 6.

[[]]

```
def depth(lst):
    “””Return the depth of the nested list.
    “””
    if not isinstance(lst, list):
        return 0.

    else:
        acc=[ 0 ]
        for s in lst:
            acc.append(depth(s).
        return max(acc) + 1.
```

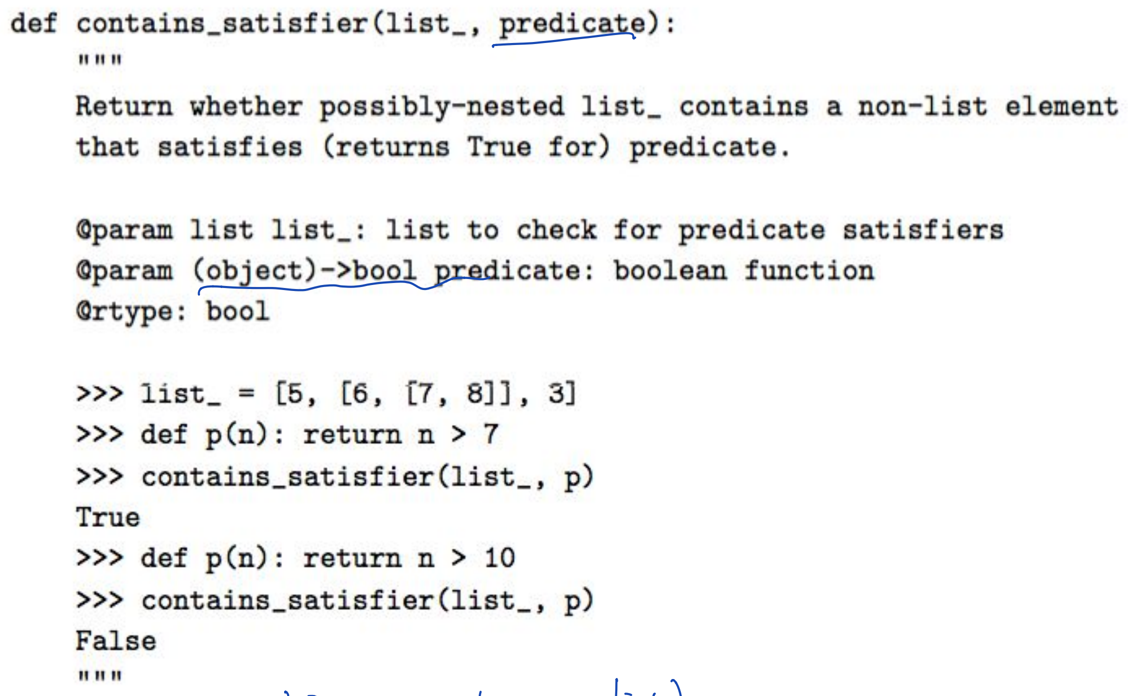


```
def count_lists(list_):
    """
    Return the number of lists, including list_ itself, contained
    in list_.

    @param list list_: list to count lists in
    @rtype: int

    >>> count_lists([])
    1
    >>> count_lists([5, [1, [2, 3], 4], 6])
    3
    """
```

[□, ☆, △]
△. △ ○

```
    if not isinstance(list_, list):
        return 0

    else:
        acc= 1.
        for s in list_:
            acc += count_lists(s)

        return acc
```

常见 Recursion 问题中 可能存在filter condition来控制base case的结果. 对于这种类型, 我们基本上可以总结成只需要在base case中进行操作



```
def contains_satisfier(list_, predicate):
    """
    Return whether possibly-nested list_ contains a non-list element
    that satisfies (returns True for) predicate.

    @param list list_: list to check for predicate satisfiers
    @param (object)->bool predicate: boolean function
    @rtype: bool

    >>> list_ = [5, [6, [7, 8]], 3]
    >>> def p(n): return n > 7
    >>> contains_satisfier(list_, p)
    True
    >>> def p(n): return n > 10
    >>> contains_satisfier(list_, p)
    False
    """
```

```
    if not isinstane(list_, list):
        return predicate(list_).

    else:
        for s in list_:
            if contains_satisfier(s, predicate):
                return True.
        return False.
```

更多**门神**喜欢的 Recursion 练习:
(要注意使用我们之前提到的三步法呦.)

Referring to the join method for str in python.
>>> str.join('+', ['a', 'b', 'c'])
'a+b+c'

Complete the following function

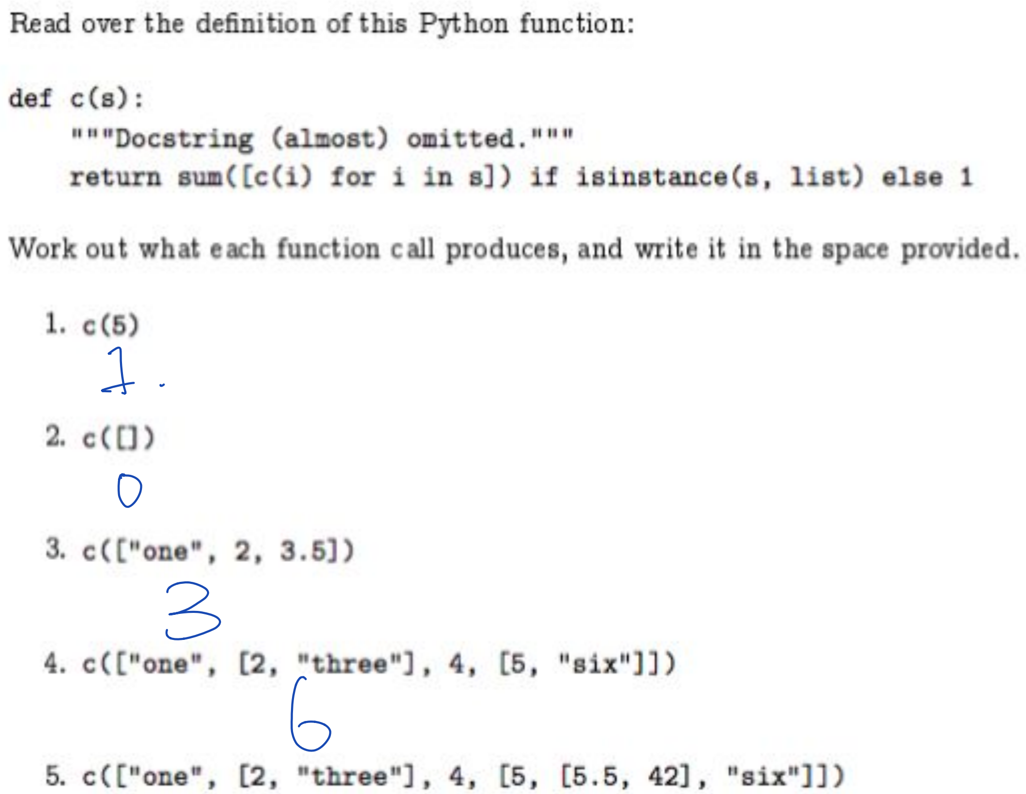
```
def nested_join(s: str, L: list) -> str:
    """ Return the join of nested list of strings L with separator s.

    >>> nested_join(' ', ['hello', ['my', 'boy']])
    'hello my boy'
    """
    if not isinstance(L, list):
        return L.
    else:
        acc = []
        for sublist in L:
            acc.append(nested_join(S, sublist))

        return str.join(S, acc)
```

```
    s  s
[□, ☆, △]
```

lab3(4)

```
def print_by_level(L):
    """ Print all items in the nested list level by level.
    """
    q = [L].
    while len(q) > 0:
        temp = q.pop(0).
        if not isinstance(temp, list):
            print(temp).
        else: q.extend(temp).
```

[[2, 3], 4, [5]]



Read over the definition of this Python function:

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

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

1. `c(5)`

   1 .

2. `c([])`

   0

3. `c(["one", 2, 3.5])`

   3

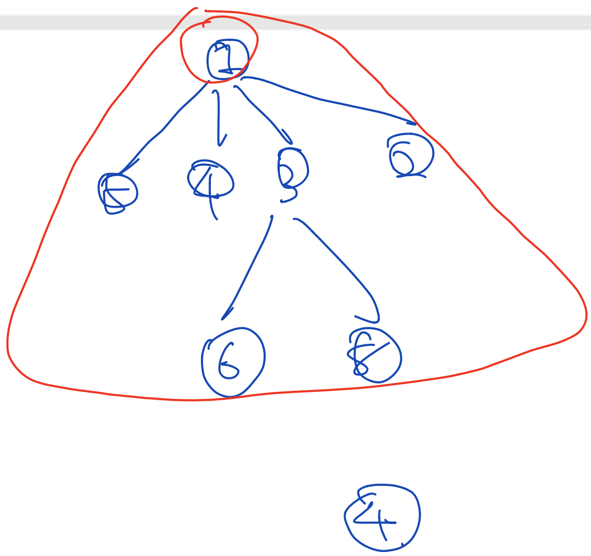
4. `c(["one", [2, "three"], 4, [5, "six"]])`

   6

5. `c(["one", [2, "three"], 4, [5, [5.5, 42], "six"]])`

   8

Section 3. Tree

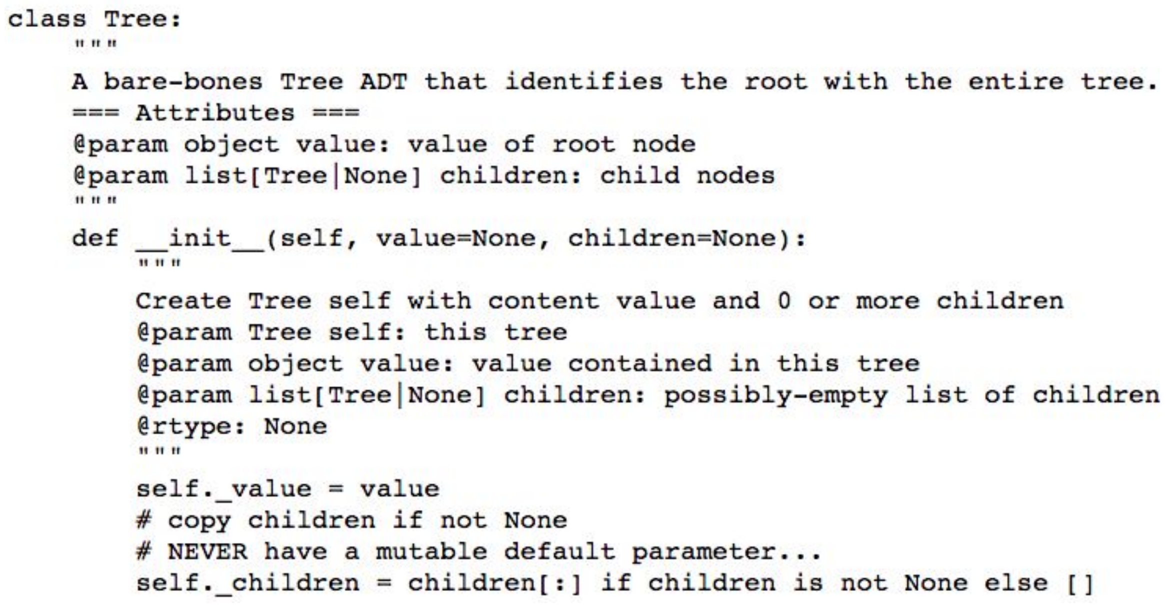


对于Tree有以下几点概念要搞清

- root
- leaf
- internal nodes

- value
- children
- height
- depth
- arity

Tree Node 的implementation



```
class Tree:
    """
    A bare-bones Tree ADT that identifies the root with the entire tree.
    === Attributes ===
    @param object value: value of root node
    @param list[Tree|None] children: child nodes
    """
    def __init__(self, value=None, children=None):
        """
        Create Tree self with content value and 0 or more children
        @param Tree self: this tree
        @param object value: value contained in this tree
        @param list[Tree|None] children: possibly-empty list of children
        @rtype: None
        """
        self._value = value
        # copy children if not None
        # NEVER have a mutable default parameter...
        self._children = children[:] if children is not None else []
```

Tree, 一个倒过来的树, 每一个tree node都有 value和children, 常见的basecase就是在没有children的时候. 如果有children, 那么要对于每个children进行recursion. 这里要注意, 类似于linkedlist, 每一个node即是一个node, 同时也代表着整个subtree.我们对一个tree object做recursion的时候, 我们可以分为两步:

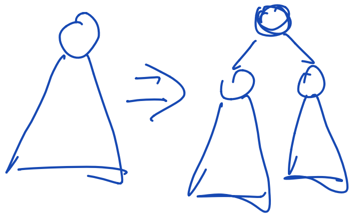
① 看自己.

1. 对node的value进行操作.
2. 对 每一个child node都 recursively 操作.

② 看 subtree.

BC: leaf

Recursive: internal



```
def gather_odd_items(t: Tree) -> list:
    “”” Return all values in the tree in a list.
    “””
    acc = []
    if t.value % 2 == 1:
        acc.append(t.value)

    for c in t.children:
        acc += gather_odd_items(c)

    return acc.
def height(t: Tree) -> int:
    “”” Return the height of the tree.
    “””
    if t.children == []:
        return 1.

    else:
        acc = []
        for c in t.children:
            acc.append(height(c))

        return max(acc) + 1.
```

```
def equivalent(t1: Tree, t2: Tree) -> bool:
    “”” Return True if the tree rooted at t1 and t2 are the same.

    >>> t1 = Tree(‘A’, [Tree(‘B’, [Tree(‘E’)]), Tree(‘C’, [Tree(‘D’)])])
    >>> t2 = Tree(‘A’, [Tree(‘B’, [Tree(‘E’)]), Tree(‘C’, [Tree(‘D’)])])
    >>> equivalent(t1, t2)
    True
    “””
    if t1.value != t2.value and len(t1.children) != len(t2.children):
        return False.

    for i in range(len(t1.children)):
        if not equivalent(t1.children[i], t2.children[i]):
            return False.

    return True..
```

```
def gather_by_depth(t: Tree) -> dict[int, list]:
    “””Gather all items in the Tree by their depth

    >>> t = Tree(‘A’, [Tree(‘B’, [Tree(‘E’)]), Tree(‘C’, [Tree(‘D’)])])
    >>> gather_by_depth(t) = {0: [‘A’], 1:[‘B’, ‘C’], 2: [‘E’, ‘D’]}
    True
    “””
    acc = {0: [t.value]}
    for c in t.children:
        temp = gather_by_depth(c)
        for depth in temp:
            if depth+1 in acc:
                acc[depth+1].extend(temp[depth])
            else:
                acc[depth+1] = temp[depth].

    return acc
```