CS 161: Fundamentals of Artificial Intelligence

Spring 2024 – Assignment 1 – Due 11:59pm, Tuesday, April 9

- Submit your commented Python program in a file named **hw1.py** via **BRUINLEARN**.
- Your programs should be written in good style. In Python, a comment is any character following a hash character (#) on a line. Provide an overall comment explaining your solutions. Furthermore, every function should have a header comment explaining precisely what its arguments are, and what value it returns in terms of its arguments. In addition, you should use meaningful variable names.
- The physical layout of the code on the page is very important for making python programs readable. Make sure that you use blank lines between functions and indent properly. Programming style will be a consideration in grading the assignment.
- You are restricted to using the python built-in functions. All other packages made by `import` (like `numpy`, `queue`, and `collections`) are forbidden.
- You may assume that all input to your functions is legal; i.e. you do not need to validate inputs.
- Do not write any additional helper functions for your code unless this is explicitly allowed. Test functions are OK.
- Your function declarations should look **exactly as specified** in this assignment. Make sure the functions are spelled correctly, take the correct number of arguments, and those arguments are in the correct order.
- Even if you are not able to implement working versions of these functions, please **include a correct skeleton** of each. Some of these assignments are auto graded and having missing functions is problematic.
- By submitting this homework, you agree to the following honor code.

  You are encouraged to work on your own in this class. If you get stuck, you may discuss the problem with up to two other students, **PROVIDED THAT YOU SUBMIT THEIR NAMES ALONG WITH YOUR ASSIGNMENT. ALL SOLUTIONS MUST BE WRITTEN UP INDEPENDENTLY, HOWEVER**. This means that you should never see another student's solution before submitting your own. You may always discuss any problem with me or the TAs. **YOU MAY NOT USE OLD SOLUTION SETS, OR COPY/USE CODE, HOMEWORK ANSWERS, OR REPORT TEXT PRODUCED BY AN AI TOOL**. Making your solutions available to other students, **EVEN INADVERTENTLY** (e.g., by keeping backups on github), is aiding academic fraud, and will be treated as a violation of this honor code.

  You are expected to subscribe to the highest standards of academic honesty. This means that every idea that is not your own must be explicitly credited to its author. Failure to do this constitutes plagiarism. Plagiarism includes using ideas, code, data, text, or analyses from any other students or individuals, or any sources other than the course notes, without crediting these sources by name. Any verbatim text that comes from another source must appear in quotes with the reference or citation immediately following. Academic dishonesty will not be tolerated in this class. Any student suspected of academic dishonesty will be reported to the Dean of Students. A typical penalty for a first plagiarism offense is suspension for one quarter. A second offense usually results in dismissal from the University of California.

1. This problem concerns the Padovan sequence. The first few elements of the sequence are 1 1 1 2 2 3 4 5 7 9 12 16 21, ... The sequence is defined as

$$\mathrm{PAD}(n+1) = \mathrm{PAD}(n-1) + \mathrm{PAD}(n-2)$$

with

$$\mathrm{PAD}(0) = \mathrm{PAD}(1) = \mathrm{PAD}(2) = 1.$$

Write a single Python function, called `PAD`, that takes a single integer argument `N`, and returns the Nth Padovan number. For example `PAD(5)` returns 3, `PAD(3)` returns 2, and `PAD(4)` returns 2.

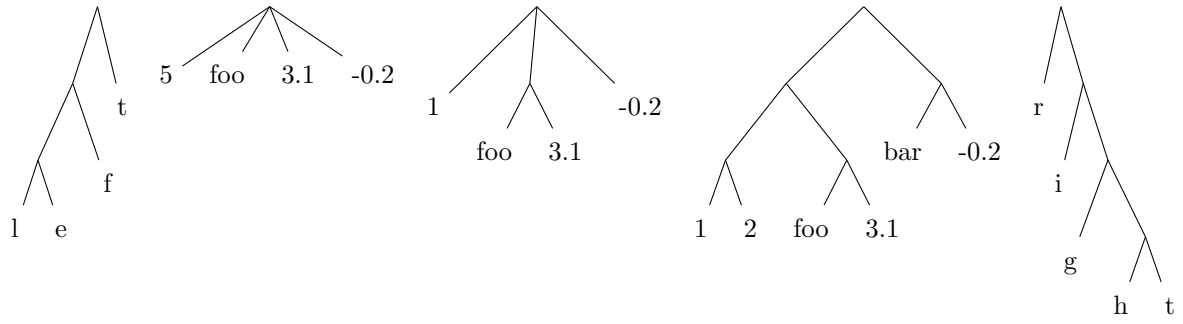
2. Write a single Python function, called `SUMS`, that takes a single numeric argument `N`, and returns the number of additions required by your `PAD` function to compute the Nth Padovan number. `SUMS` should not call `PAD`, but rather you should design the recursion for `SUMS` by examining your `PAD` code.

3. A tree can be represented in *tuples* in Python as follows:

(a) if the tree contains a single leaf node L, it can be represented by a *non-tuple object* `L`

(b) if the tree has more than one node and is rooted at N, then it can be represented by a tuple `(S1, S2, ... Sk)` where `Si` represents the ith subtree of N.

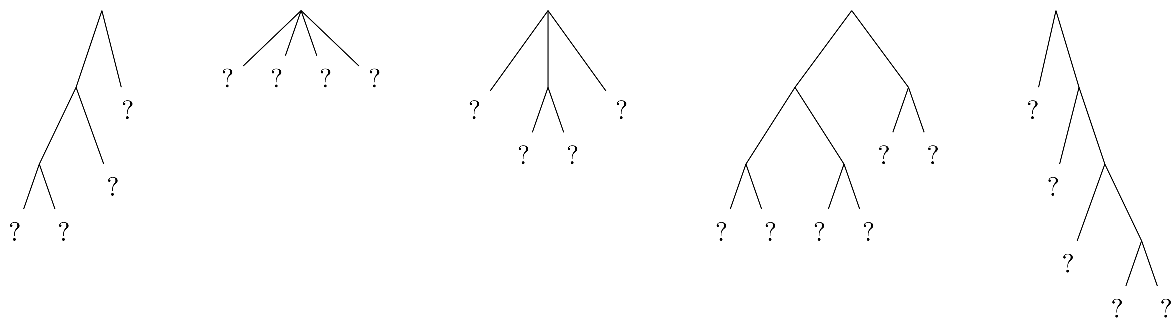
Consider for example the following five trees.



Their representations in tuples are respectively

```
(((“l”, “e”), “f”), “t”),
(5, “foo”, 3.1, -0.2),
(1, (“foo”, 3.1), -0.2),
(((1, 2), (“foo”, 3.1)), (“bar”, -0.2))
(“r”, (“i”, (“g”, (“h”, “t”)))).
```

Write a single Python function, called `ANON`. It takes a single argument `TREE` that represents a tree, and returns an anonymized tree with the same structure, but where every leaf in the tree is replaced by a question mark. The anonymized versions of the trees above are as follows.



Test your program on at least these inputs:

```
>>> ANON(42)
’?’
>>> ANON("FOO")
’?’
>>> ANON(((("L", "E"), "F"), "T"))
(((‘?’, ’?’), ’?’), ’?’)
>>> ANON((5, "FOO", 3.1, -0.2))
(’?’, ’?’, ’?’, ’?’)
>>> ANON((1, ("FOO", 3.1), -0.2))
(’?’, (’?’, ’?’), ’?’)
>>> ANON((((1, 2), ("FOO", 3.1)), ("BAR", -0.2)))
(((’?’, ’?’), (’?’, ’?’)), (’?’, ’?’))
>>> ANON(("R", ("I", ("G", ("H", "T")))))
(’?’, (’?’, (’?’, (’?’, ’?’))))
```

Hint: use `type(X) is tuple` to check if the input `X` is a tuple or not

4. Write a single Python function, called `TREE_HEIGHT`, which takes a tree `TREE` in *tuples* (same definition as Question 3), and returns the height of `TREE`. Note that the height of a tree is defined as the length of the longest path from the root node to the farthest leaf node.

```
>>> TREE_HEIGHT(1)
0
>>> TREE_HEIGHT((5, "FOO", 3.1, -0.2))
1
>>> TREE_HEIGHT((1, ("FOO", 3.1), -0.2))
2
>>> TREE_HEIGHT(("R", ("I", ("G", ("H", "T")))))
4
```

5. An *ordered tree* is either a number n or a tuple (L, m, R), where

- L and R are ordered trees
- m is a number
- all numbers appearing in L are smaller than m
- all numbers appearing in R are larger than m

Write a single Python function, called `TREE_ORDER`, which takes one argument `TREE` that represents an *ordered tree*, and returns a tuple that represents the postorder traversal of the numbers in `TREE`.

```
>>> TREE_ORDER(42)
(42,)
>>> TREE_ORDER(((1, 2, 3), 7, 8))
(1, 3, 2, 8, 7)
>>> TREE_ORDER(((3, 7, 10), 15, ((16, 18, 20), 30, 100)))
(3, 10, 7, 16, 20, 18, 100, 30, 15)
```