Week 10: Agenda

- Term Project
- Advanced Recursion:
  - CT
  - Tree Recursion
  - More examples
- OOP

```
def ct(a, b, d=0):
    print(f"a:{a} b:{b} d:{d}")
    n = b - a
    if n <= 1:
        return
    ct(a, a+n//2, d+1)
    ct(a+n//2, b, d+1)

ct(0,3)
```

Term Project: Important Dates

Term Project Agreement Form

TP0

Mon 24-Mar

- Final TP idea
- Meeting with your mentor

TP1

Tue 9-Apr, 6pm

- Design proposal (document) and preliminary code
- Meeting with your mentor

TP2

Tue 15-Apr, 6pm

- Working demo (code)
- Meeting with your mentor

TP3

Tue 22-Apr, 4pm

- Final code
- Video
- Demo to your mentor

Unlocking the power of recursion

Tree Recursion: When you make a recursive call more than once in your recursive case

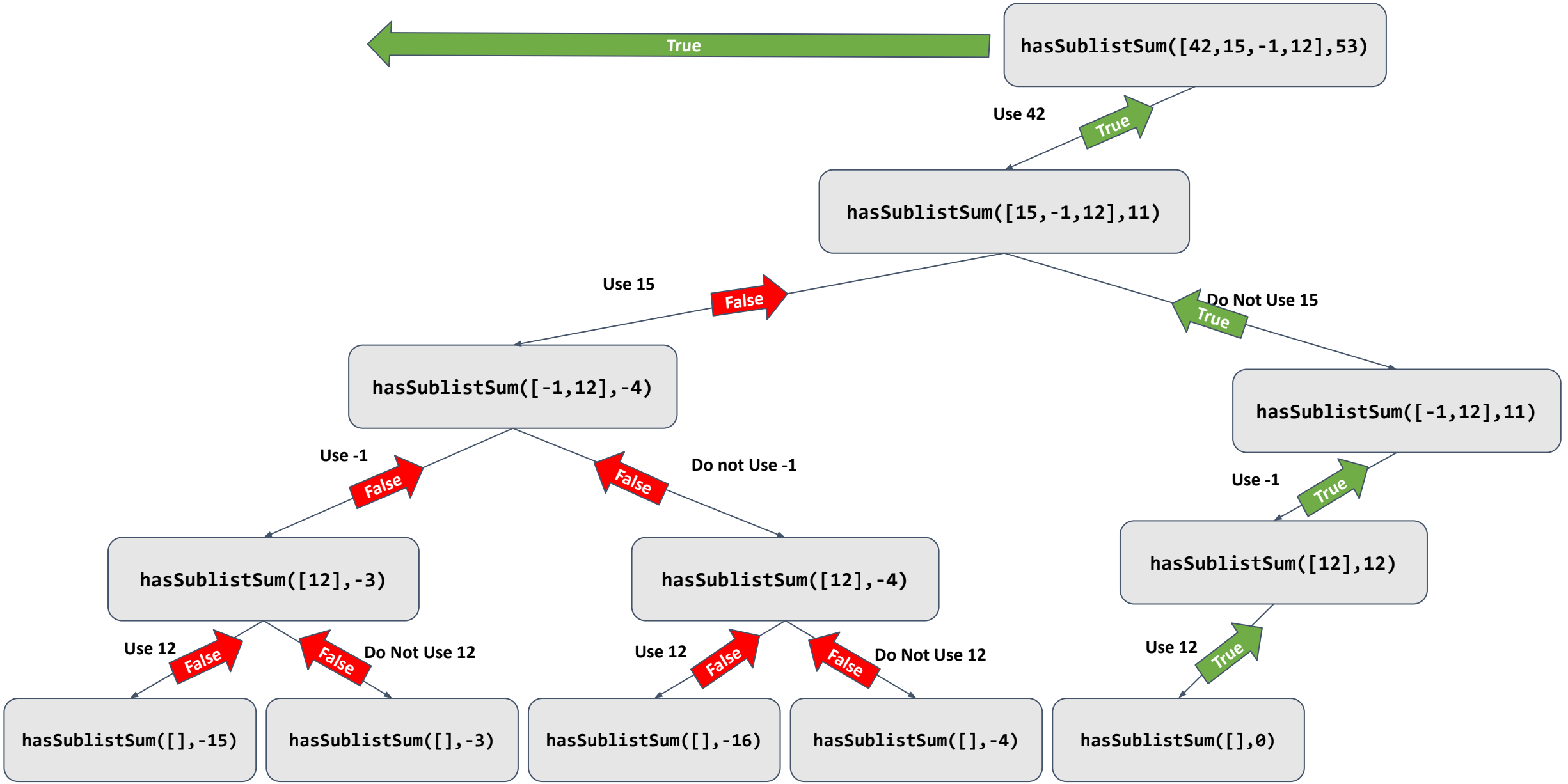
Why?

- Some problems are more easily solved by tree recursion.
- Brute forcing solutions (Backtracking)

Some Examples: `hasSublistSum`

`hasSublistSum(L, s)`

Write the function `hasSublistSum(L, s)` that takes a list of integers L and an integer s, and returns `True` if there exist elements in L that sum to s. Otherwise, the function returns `False`.


True
hasSublistSum([42,15,-1,12],53)
Use 42
True
hasSublistSum([15,-1,12],11)
Use 15
False
Do Not Use 15
True
hasSublistSum([-1,12],-4)
hasSublistSum([-1,12],11)
Use -1
False
Do not Use -1
False
Use -1
True
hasSublistSum([12],-3)
hasSublistSum([12],-4)
hasSublistSum([12],12)
Use 12
False
False
Do Not Use 12
Use 12
False
False
Do Not Use 12
Use 12
True
hasSublistSum([],-15)
hasSublistSum([],-3)
hasSublistSum([],-16)
hasSublistSum([],-4)
hasSublistSum([],0)

Example:

Write the function `coinChange(coins, amount)` that calculates the number of unique ways to make change for the given amount using the coins. You can use each coin an unlimited number of times.

Can you return the ways to make change as a list of lists?

E.g.,

`coinChange([1,2,5], 5)` returns `[[5],[2,2,1],[2,1,1,1],[1,1,1,1,1]]`

Example A: **`getHiLo(L)`**

- Write the function `getHiLo(L)` that receives a list of integers `L` and returns a tuple `(a,b)` where `a` is the lowest number and `b` is the highest. You can assume `len(L) > 0`
- Examples:
  - `getHiLo([1,2,3,4,5]) == (1,5)`
  - `getHiLo([42,4,5,-6]) == (-6,42)`
  - `getHiLo([42]) == (42,42)`

Example B: `indexMap(L)`

- Write the function `indexMap(L)` that takes a 1D list L and returns a dictionary that maps each value in L to a set of the indices in L where that value occurs. For example:
- `indexMap([5, 6, 5]) == { 5:{0,2}, 6:{1} }`
- `indexMap([9, 6, 3, 6, 9]) == { 3:{2}, 6:{1,3}, 9:{0,4} }`

Midterm 2 topics end here

Term Project

- What? The minimum
  - You should produce an interactive Python application.
  - Sufficiently complex
  - Using cmu_graphics
- Beyond
  - Something that shows algorithmic complexity (AI, etc)
  - An amazing user experience. (The look and feel, graphics, etc.)
  - The usage of an external module to add a feature. (Multi-player, fancy graphics, etc.)

Object Oriented Programming

aka OOP

Definitions

- Function
- Class = Function(s) + Data (declarative)
  - Functions inside a class are call methods
  - Data are called properties
- Example:
  - strings:
    - Data: sequence of characters
    - Functions: `replace, find, split`, …
  - lists:
    - Data: sequence of elements
    - Functions: `append, pop, find, index, …`

Creating your own `class`

Example: `Circle`

- Properties?
  - What data define a circle?
- Methods?
  - How do you plan to use circles in your code?
    - `.getArea()`
    - `.getDiameter()`
    - `.doIntersect(anotherCirle)`

Some special methods

- `__init__`: constructor, initializer
  - Define initial values for the properties when an object is created
  - It does NOT return any meaningful value
- `__repr__`:
  - Returns a string
  - Python uses it to convert an object to a string
  - E.g., `print`
- `__eq__`: comparator
  - Returns True if the object is equal to another object
  - Python uses it for testing equality

What you need to know

- The idea behind OOP
- Class vs Object
- Functions vs Methods
- How to define a class
- How to use a class