CS 30 Discussion 1A

2020.10.16

Welcome back to CS30 Discussion!

- Some administrative things:
  - Please join Piazza for help :)
  - HW 1 Solutions will be released by **Monday morning**.
    - Because of late days, we can't review them in discussion section until next week, but we'll make sure to go over them before the first midterm
  - HW 2 has been posted on CCLE

Common Mistakes In Submitting HW#1 and How To Prevent Them

- READ the instructions on how to submit. E.g. For HW #1 you need to submit three different files for picobot and statistics
- DO NOT change function names
- DO NOT change file names or file type (e.g. you are asked to submit .py file)
- Submit only the files you're requested to submit. If you're given problem prompts in a single file, fill them out in that file; do not create a new file for each function.
- Failing to follow instruction will result in **0** from HW #2.

Syntax Review: Boolean Expressions in Conditions

Remember that for our purposes:

- an `if` is followed by a condition
- a condition evaluates to either `True` or `False`
- we need `thing1` [some operator] `thing2` to make a complete *expression*
- we can also use `and`, `or` to combine *expressions*, but remember that we need a complete *expression* one either side
    - Example: `if x > 7 or x == 3` ✔️
    - `if x > 7 or == 3` ⛔
    - `if x == 3 or 2` ⛔ `(we let x = 2, True/False?)`

Warm Up Question

How would you translate the following into code? (Try not to use IDLE for these questions! Write them into Text Editor first and then transfer them over.)

- a condition that evaluates to `True` when `x` is a positive number and is divisible by 3 and 4.
  - x > 0 and x % 3 == 0 and x % 4 == 0
    x > 0 and x % 12 == 0
- a condition that tells you whether a character in variable `x` is a vowel or not
  - `x == 'a' or x == 'e' or x== 'i' or x == 'u' or x == 'o'`
  - or, `x in 'aeiou'` in Python only
- an expression that evaluates to `True` for all numbers that are even except the number `6`:
  - `x % 2 == 0 and x != 6   # the % operator yields the remainder from the division of the first argument by second`

Warm Up Question

```
work_authority (10)
work_authority (40)
work_authority (70)
work_authority (-5)
work_authority (“too old”)
```

```
def work_authority (age):
    if age < 18 and age >= 0:
        return " You are Minor. You are not Eligible to Work "
    else:
        if age >= 18 and age <= 60:
            return " You are Eligible to Work. Please fill in your details and
    apply "
        elif age > 60:
            return " You are too old to work as per the Government rules. Please
    Collect your pension!"
        else:
            return " wrong input! "
```

Concept Review: List and String

- **Indexing:**
  The indexing operator ([ ]) selects a single element from a list/string. The expression inside brackets is called the index, and must be an integer value. The index indicates which element to select, hence its name.
  e.g. x = [‘hi’,2,4]
  x [1] = ?
- **Length:**
  the **len** function is used to get the number of values in a list/string. E.g.
  - Accessing elements at the end of a sequence
    Test = [1,2,3]; Test[len(Test)] Wrong!!!

Concept Review: List and String

- **Slicing:**

The slicing operator [startIndex:endIndex) selects elements within the start and end index from a list/string. Start and end index should be Integer values. e.g. x = [‘hi’,2,4,‘Michelle’,‘Doe’]

x [1:4] = ?

If-elif-else Syntax

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
```

Remember: if-elif-else implies that each "case" is *exclusive*.

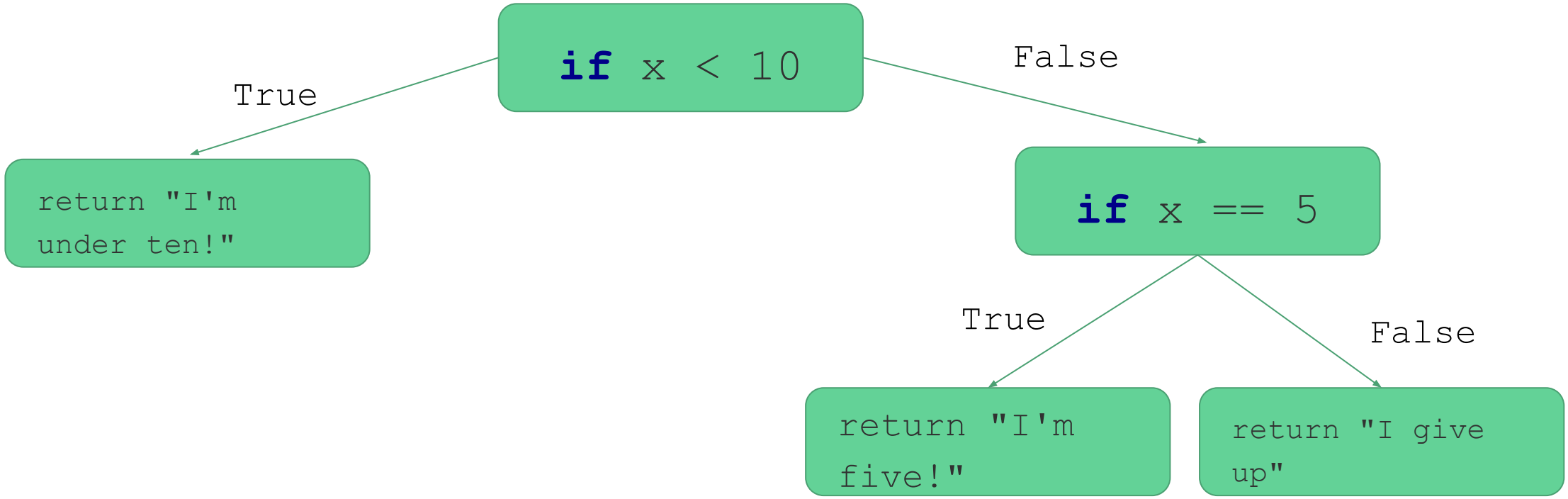
What would the following print out if x were 5?

```
if x < 10:
    return "I'm under ten!"
elif x == 5:
    return "I'm five!"
else:
    return "I give up"
```

How to trace a if-else condition (Flowcharting!)

```
if x < 10:
    return "I'm under ten!"
elif x == 5:
  return "I'm five!"
else:
  return "I give up"
```

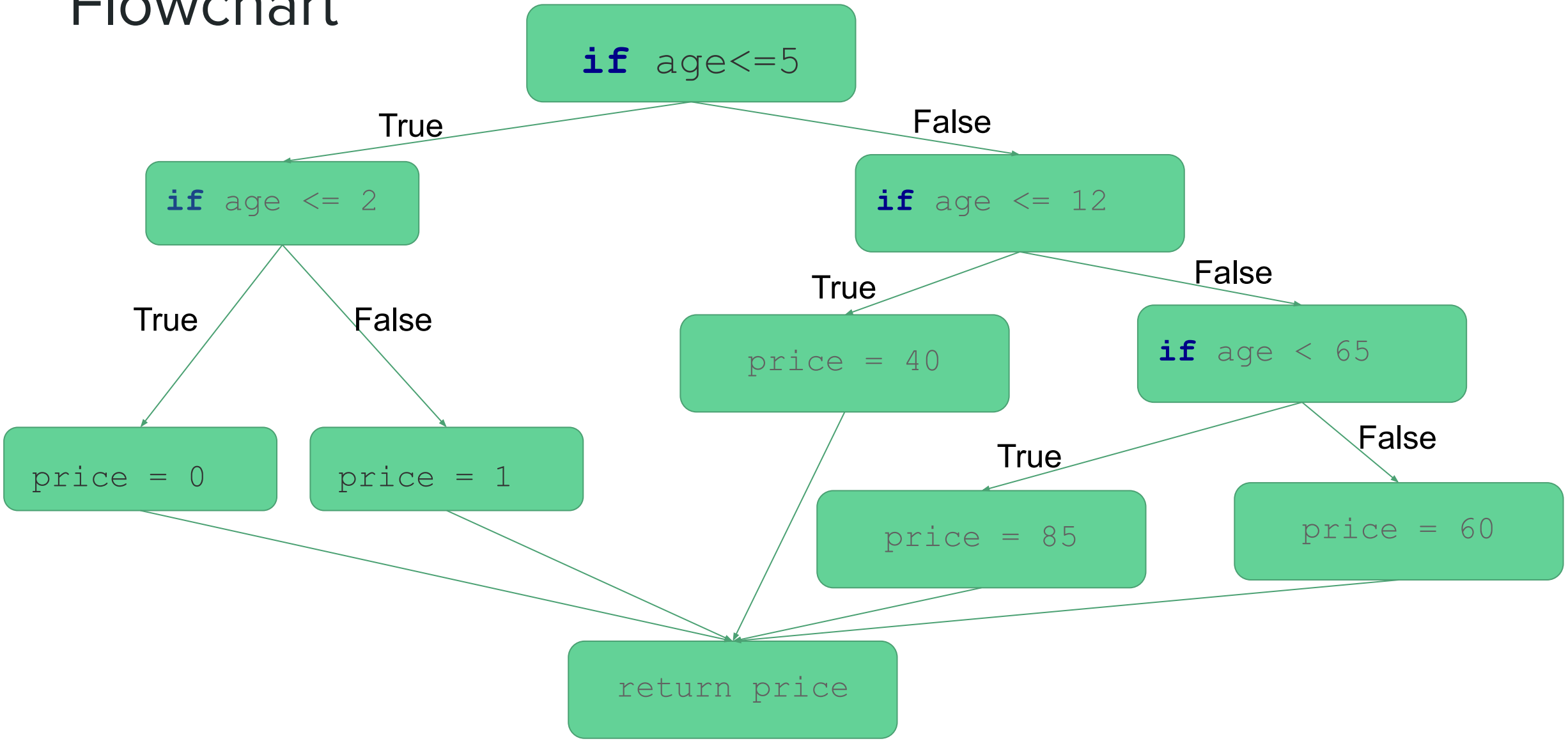
Flowchart


if x < 10
True
False
return "I'm under ten!"
if x == 5
True
False
return "I'm five!"
return "I give up"

Flowcharting practice

```
def getPrice(age):
    if age <= 5:
        if age <=2 :
            price = 0
        else:
            price = 1
    elif age <= 12:
        price = 40
    elif age < 65:
        price = 85
    else:
        price = 60
    return price
```

Flowchart


if age<=5
True
False
if age <= 2
if age <= 12
True
False
True
False
price = 40
if age < 65
price = 0
price = 1
True
False
price = 85
price = 60
return price

Good god Lemon, spot the mistakes

```
def getPrice([age]):
    if age <= 2:
        price = 0
    elif age[2] = 12:
        price = 40
    elif age[1] < 65 and > 25:
        price = 85
        else:
            price = 60
    return price
```

```
>>> ageLst = [1,2,3]
>>> getPrice(ageLst)
```

Mistakes

```
>>> ageLst = [1,2,3]
>>> getPrice(ageLst)
def getPrice([age]):
    price = 0
     if age <= 2:
         price = 0
     elif age[2] = 12:
         price = 40
     elif age[1] < 65 and > 25:
         price = 85
         else:
             price = 60
     return price
```

Recursion Review

Remember that recursive functions are just functions that call themselves.

You first **call yourself recursively on a slightly smaller version** of the argument, before doing anything else.
Then the key is that you get to **assume that the recursive call does the right thing**, and now your job is to figure out **how to use that result to produce the overall result** that is desired.

Recursion Review

If you're stuck, try answering the following questions for this problem:

1. How can I break this into smaller pieces?
2. **What are the base cases?** (When are some situations I have enough information to give a definite answer to this question?)
3. **What *operations* do I have to do repeatedly** to answer this question? (recursive body)

Recursion Practice

Read the following code. What does it do? How many times is this function called, and with what arguments? Hint: try expanding the last statement with each new function call!

```
# assume n > 0
def foo(n):
    if n == 1:
        return 0
    elif n == 2:
        return 1
    else:
        return foo(n - 1) + foo(n - 2)
>> foo(4)
```

Recursion Practice

```
foo(4)
   foo(3) + foo(2)
      foo(2) + foo(1)
      1 + foo(2)
   1 + 1
2
```

Recursion Practice

Step 1: the first function call, foo(4), skips down to the else branch. That generates two more calls: foo(3) and foo(2). The return expression is "return foo(3) + foo(2)”

Step 2: foo(3) is evaluated next. It also skips down to the else branch, generating two more calls: foo(2) and foo(1). foo(3) returns "foo(2) + foo(1)" which means the original return expression can be rewritten as foo(2) + foo(1) + foo(2).

Step 3: foo(2) takes the elif branch, returning 1. we can rewrite the above expression as 1 + foo(1) + 1.

Step 4: foo(1) takes the if branch, so we can finally rewrite the expression as 1 + 0 + 1 giving us 2. That's 5 calls total: foo(4), foo(3), foo(2), foo(1) , and foo(2)

Recursion Practice: Once More With Feeling

What does this function return when x=36 and y=12? Can you come with a general description of what it does?

```
def mystery(x, y):
   if x < y:
      return mystery(x, y-x)

    elif y < x:
      return mystery(x-y, x):

    else:
      return x
```

Recursion Practice

Given a list of integers, e.g. [3, 1, 2, 10, 7, 5], return the list modified so that it contains only even integers.

Recursion Practice

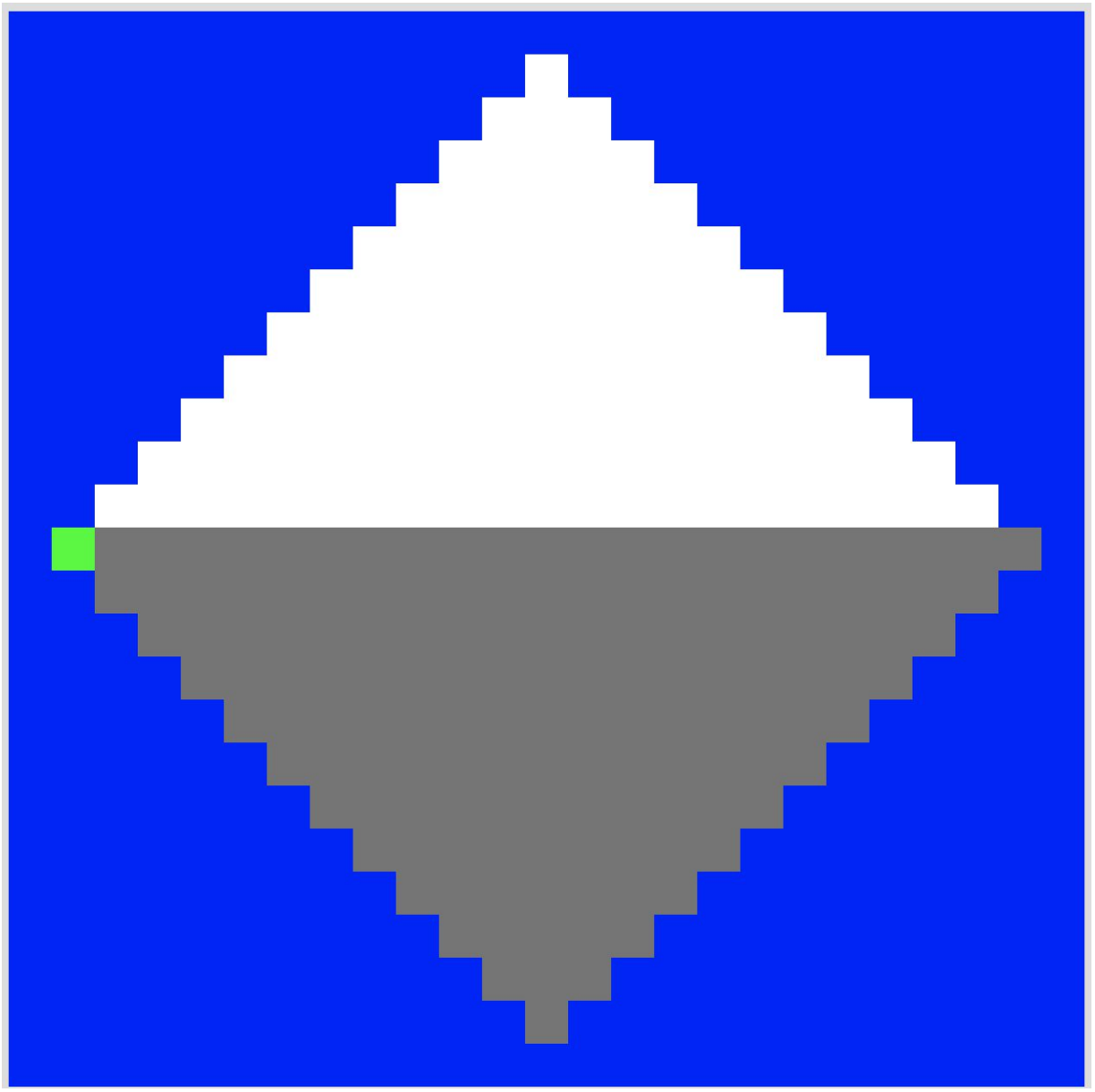
```
def even(l):
    if l == []:
        return []
    else:
        head = l[0]
        tail = l[1:]  # on a size 1 list, returns empty list
        evenRest = even(tail)
        if head % 2 == 0:
          return [head] + evenRest
        else:
          return evenRest
```

Picobot

```
# Go to the most South point
0 ***x -> S 0
0 xExS -> W 0
0 xxWS -> E 0
0 xEWS -> N 1

# Head to East and West
1 *x** -> E 1
1 *E** -> W 2
2 **x* -> W 2

# Head to North
2 x*W* -> N 1
```



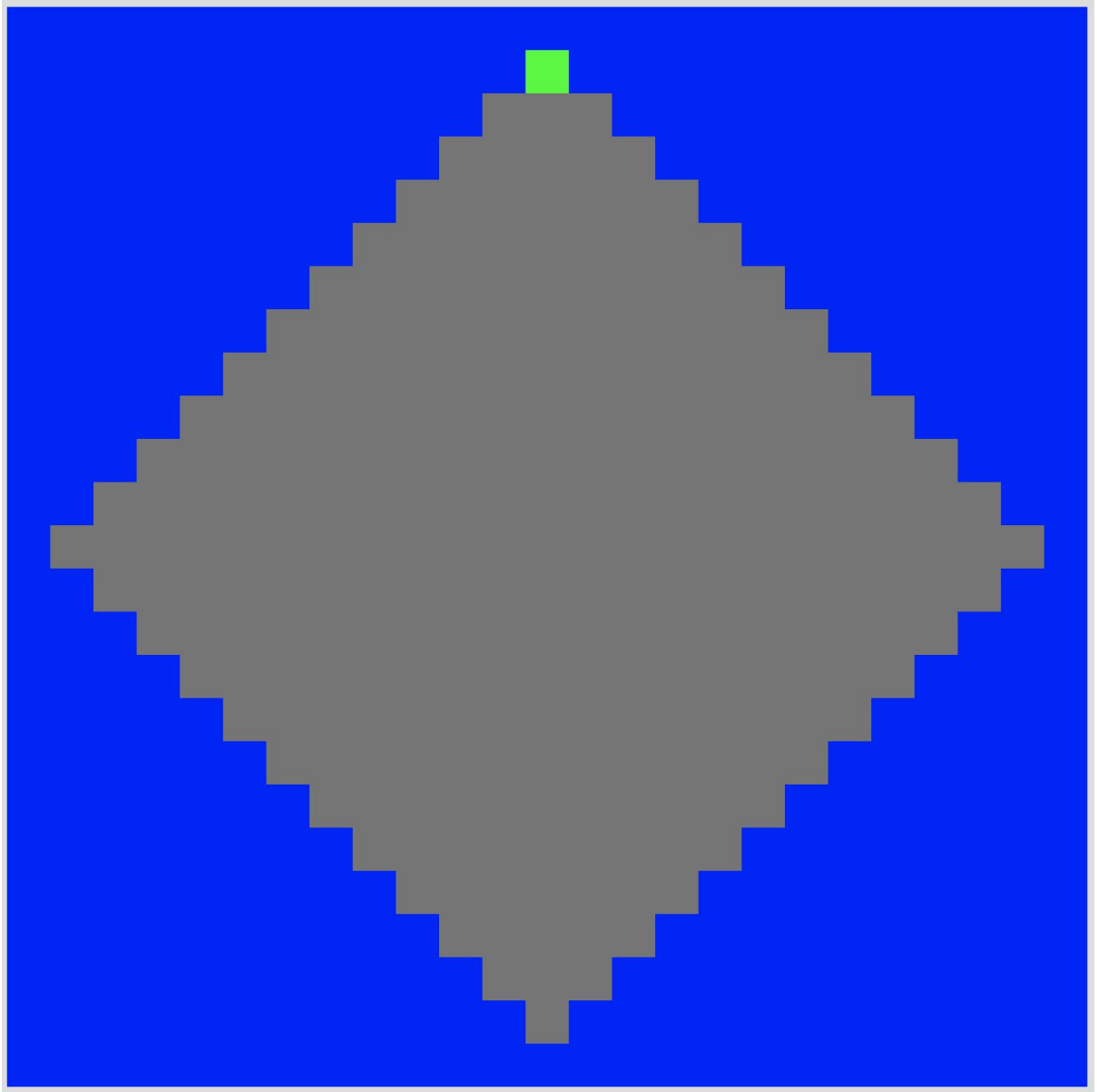
Picobot

```
# Go to the most South point
0 ***x -> S 0
0 xExS -> W 0
0 xxWS -> E 0
0 xEWS -> N 1

# Head to East and West
1 *x** -> E 1
1 *E** -> W 2
2 **x* -> W 2

# Avoid trap
2 **W* -> E 3

# Head to North
3 x*** -> N 1
```



WorkSheet