The deadline for this exercise sheet is **Monday, 23.04.2018, 8:00.**

1 Boolean Operators

Determine the truth values of the following boolean operators for each configuration of truth values as given in the tables

1.1

a	b	c	(b or c) and (a or c)
true	true	true	**Solution:** true
true	true	false	**Solution:** true
true	false	true	**Solution:** true
true	false	false	**Solution:** false
false	true	true	**Solution:** true
false	true	false	**Solution:** false
false	false	true	**Solution:** true
false	false	false	**Solution:** false

1.2

a	b	c	a or (b and c) and (not c or not a)
true	true	true	**Solution:** true
true	true	false	**Solution:** true
true	false	true	**Solution:** true
true	false	false	**Solution:** true
false	true	true	**Solution:** true
false	true	false	**Solution:** false
false	false	true	**Solution:** false
false	false	false	**Solution:** false

1.3

a	b	c	not(not(b and not(c or a)))
true	true	true	**Solution:** false
true	true	false	**Solution:** false
true	false	true	**Solution:** false
true	false	false	**Solution:** false
false	true	true	**Solution:** false
false	true	false	**Solution:** true
false	false	true	**Solution:** false
false	false	false	**Solution:** false

2 Warm up – Prof Strikes Again

Remember the task from last week? Now that your prof computed your numerical grade, the functionality of this program shall be extended by automatically determining whether someone passed the class or not based on this grade.
Write a function `passed` that takes a grade as a parameter and returns whether the student passed or failed.

Solution:

```
1 def passed(grade):
2   """Returns True if the grade is a passing one"""
3   return grade <= 4.0
```

3 Loops

3.1 N Bottles of Beer

Similar to Hello World programs, 99 bottles programs give us an idea of how a programming language looks as they show the basic loop concepts. The 99 bottles program sings a little song which goes like this:

99 bottles of beer on the wall, 99 bottles of beer. Take one down and pass it around, 98 bottles of beer on the wall.
98 bottles of beer on the wall, 98 bottles of beer. Take one down and pass it around, 97 bottles of beer on the wall.
. . .
1 bottle of beer on the wall, 1 bottle of beer. Take one down and pass it around, no more bottles of beer on the wall.

Write a function `n_bottles(n)` in the script `n_bottles.py` which sings the song starting with n bottles instead of 99. If n is bigger than 99 or smaller than 5 print a message that you want to sing a funnier song than n bottles (of course replace n with the current n). Your final result may also structure the verses in a different layout.

Solution:

```
1  def bottles(n):
2      """Formats plural according to number of bottles."""
3      return ('1 bottle' if n == 1 else str(n) + ' bottles') + ' of beer'
4
5  def n_bottles(n):
6      """
7      Go through the bottles of beer on the wall, and pass them around.
8
9      Calls `bottles(n)` for formatting, and reduces n by 1 each
           iteration
10     """
11     if 5 <= n <= 99:
12         while(n > 0):
```

```
13              print(bottles(n) + ' on the wall,\n  ' + bottles(n) + '.')
14              n = n - 1
15              print('Take one down and pass it around,\n  ' +
16                  # conditional expression to determine whethere there
                        are bottles left
17                  bottles(n if n > 0 else 'no more') +
18                  ' on the wall.\n')
19      else:
20          print('I want to sing funnier songs than "' + bottles(n) + '".\
                n')
21
22
23  n_bottles(3)
24  n_bottles(1011)
25  n_bottles(5)
```

Output:

```
I want to sing funnier songs than "3 bottles of beer".

I want to sing funnier songs than "1011 bottles of beer".

5 bottles of beer on the wall,
   5 bottles of beer.
Take one down and pass it around,
   4 bottles of beer on the wall.

4 bottles of beer on the wall,
   4 bottles of beer.
Take one down and pass it around,
   3 bottles of beer on the wall.

3 bottles of beer on the wall,
   3 bottles of beer.
Take one down and pass it around,
   2 bottles of beer on the wall.

2 bottles of beer on the wall,
   2 bottles of beer.
Take one down and pass it around,
   1 bottle of beer on the wall.

1 bottle of beer on the wall,
   1 bottle of beer.
Take one down and pass it around,
   no more bottles of beer on the wall.
```

3.2 Return of the turtle

You hopefully remember the turtle from the first week – you drew a Saint Nicholas' house with it. This time we will draw even more houses! And some trees to keep them company.
Write a script `turtle_world.py`. Develop a function `draw_house` that draws a house – it doesn't need to be the Saint Nicholas' house, but you can recylce your code if you like. Then write a function `draw_treeheight` that draws a tree of a given "height". The height serves a double purpose as the branching factor of the tree. Your tree is going to be a fractal, which means that it will be a pattern that repeats itself recursively.

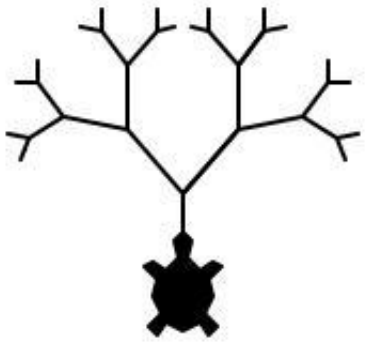

Figure 1: Example Tree

To build the tree follow this algorithm:

```
To draw a tree with height h:
  If height h is 0, stop.
  Draw a line of length L * h.
  Rotate left by angle A.
  Draw a tree of height h - 1.
  Rotate right by angle 2A.
  Draw a tree of height h - 1.
  Rotate left by angle A.
  Move back to the beginning of the line.
```

The resulting tree should look similar to Figure 1. Choose A and L as you like, be creative!

Now draw a simple landscape. Draw a house, a small tree, a big tree, a small tree, and repeat this pattern a couple of times (Figure 2). Or build a different one. Just make it repetitive (Yes, use loops).

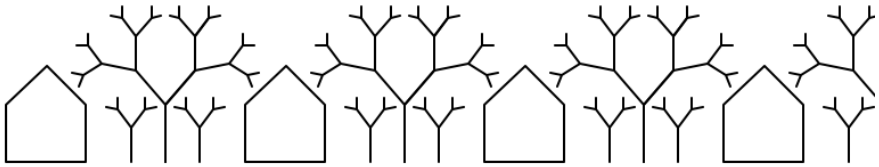

Figure 2: Flat World

Bonus: Can you build a round world like in Figure 3?

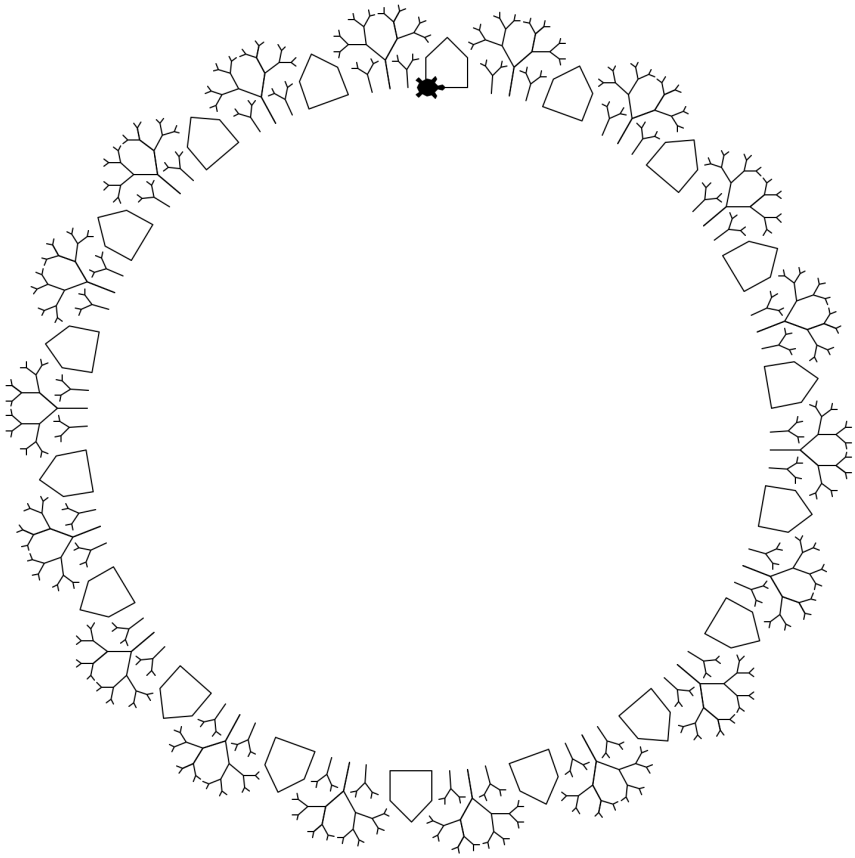

Figure 3: Round World

Solution:

```
1  # pylint: disable=E1101
2  import time
3  import turtle
4
5
6  # constants for all draw functions for a nice and consistent look
7  LENGTH = 6
8  ANGLE = 35
9
10 def draw_tree(height):
11     """
12     Draws a fractal tree with `height` repetitions.
13
14     `height` defines the overall height of the tree is also responsible
           for
```

```
15      the length of each branch in every iteration
16      """
17      if height == 0:
18          return
19      # left branch
20      turtle.forward(LENGTH * height)
21      turtle.left(ANGLE)
22      draw_tree(height - 1) # drawing the little tree at the end of the
            branch
23      # right branch
24      turtle.right(2 * ANGLE)
25      draw_tree(height - 1)  # drawing the little tree at the end of the
            branch
26      turtle.left(ANGLE)
27      turtle.backward(LENGTH * height)
28
29
30  def draw_house():
31      """This draws a nice and simple house!"""
32      # the dimensions of our house
33      height = 5 * LENGTH
34      width = 7 * LENGTH
35      roofside = (width ** 2 / 2) ** (1 / 2)
36
37      # left wall
38      turtle.forward(height)
39      # roof
40      turtle.right(45)
41      turtle.forward(roofside)
42      turtle.right(90)
43      turtle.forward(roofside)
44      turtle.right(45)
45      # right wall
46      turtle.forward(height)
47      turtle.right(90)
48      # bottom line
49      turtle.forward(width)
50      turtle.right(90)
51
52
53  def draw_world(curvature_step=0):
54      """
55      This draws a turtle world.
56
57      The curvature step is relevant for drawing a round world.
58      The higher the curvature step is, the smaller our circle will be.
59
60      Each village will consist of one house and 3 trees, with one being
            taller.
61      """
62      if curvature_step > 0: # this ensures we are going full circle
63          villages = 360 // 4 // curvature_step
64      else: # 5 villages for our flat world
65          villages = 5
66
67      # the _ is called an anonymous variable, since we don't use it
            anyway
```

```
68      # we don't need to give it a name. It just acts as a counter.
69      for _ in range(villages):
70          prepare_drawing()
71          draw_house()
72          finish_drawing()
73
74          turtle.right(curvature_step)
75          turtle.forward(LENGTH * 11)
76
77          # and draw the three trees
78          for j in range(3):
79              prepare_drawing()
80              # the middle one will be 5 high, since we iterate over
                    0,1,2
81              # and only for 1 modulo 2 is 1 returned.
82              draw_tree(3 + j % 2 * 2)
83              finish_drawing()
84
85              turtle.right(curvature_step)
86              turtle.forward(LENGTH * 3)
87
88          turtle.forward(LENGTH)
89
90
91  def init():
92      """set up the turtle parameters"""
93      turtle.reset()
94      turtle.shape('turtle')
95      turtle.speed('fastest')
96      turtle.up()
97
98
99  def prepare_drawing():
100     """move the pen down to actually draw and make turtle upright"""
101     turtle.down()
102     turtle.left(90)
103
104
105 def finish_drawing():
106     """move pen up to stop drawing and return turtle to axis"""
107     turtle.right(90)
108     turtle.up()
109
110
111 def draw_flat_world():
112     """wrapper to start drawing a flat world with 0 curvature"""
113     init()
114     turtle.goto(-300, 0)
115
116     draw_world()
117     turtle.goto(0,0)
118
119
120 def draw_round_world(curvature_step=5):
121     """wrapper to draw a curved world with a default curvature step of
            5"""
122     init()
```

```
123     turtle.goto(0, 300)
124     draw_world(curvature_step)
125     turtle.goto(0,0)
126
127
128 def draw():
129     """Draw the flat world. Rest shortly to marvel at it. Draw round
            world."""
130     draw_flat_world()
131     time.sleep(3)
132     draw_round_world()
133     turtle.done()
134
135 # Start the party!
136 draw()
```