Computational Thinking

Practical Week 2

Exercises

1. Add parentheses to each expression so that it evaluates to `True`:

   (a) `0 == 1 == 2`

   (b) `2 + 3 == 4 + 5 == 7`

   (c) `3 < 4 and 5 < 4 or 1 < 2 and not 1 < 1`

   (d) `1 < 2 and 2 < 1 or not 2 < 4 and 3 < 1`

2. A dartboard of radius 10 and the wall it is hanging on are represented using the two-dimensional coordinate system, with the board's centre at coordinate $(0, 0)$. Variables `x` and `y` store the x- and y-coordinate of a dart hit. Write an expression using variables `x` and `y` that evaluates to `True` if the dart hits the dartboard, and evaluate this expression for the dart coordinates $(0, 0)$, $(10, 10)$, $(-6, 8)$ and $(-7, 8)$.

3. Type:

   ```
   >>> s0 = '0'
   >>> s1 = '1'
   ```

   Now write expressions using only `s0`, `s1` and Python string and algebraic operators that evaluate to each of the following strings:

   (a) `'01'`

   (b) `'0101'`

   (c) `'110110'`

   (d) `'0110110110110'`

   (e) `'10011001100110110'`

   (f) `'11010010010110100100101101001001 0'`

   You should try to minimise the length of your expressions.

4. Type:

```
>>> x = 'thequickbrownfox jumpsoverthelazydog'
```

   Write expressions, using just `x` and Python string and algebraic operators, that evaluate to `'a'`, `'z'` and `'iain is cool'`.

5. Type:

```
>>> x = 'hello'
>>> y = 'goodbye'
```

   Write Boolean expressions, using just `x`, `y`, the built-in function `len()` and Python boolean, string and algebraic operators, that check whether:

   (a) `x` and `y` have the same lengths
   (b) the third characters of `x` and `y` are the same (remember, the 3rd character of `x`, say, is `x[2]`)
   (c) either `x` or `y` has two consecutive l's
   (d) the first character of `x` is not in `y`
   (e) `x` comes before `y` in the lexicographic ordering (that is, the 'telephone directory' ordering)
   (f) the length of `x` plus the length of `y` is odd (don't use any numbers in your expression!).

   Try your expressions on two different strings `x` and `y` (but of the same lengths, 5 and 7, respectively).

6. Type:

```
>>> help(str)
```

   Use the `find` method to take a string `x` that is known to hold the name of a person, with the first name separated from the second name by a space, and output a string of length 2 consisting of the initials of this person. Try your expression on some samples.

7. The range of a list of numbers is the largest number in the list minus the smallest number. Write an expression to give the range of any list of numbers and test it out.

8. Type:

```
>>> help(list)
>>> a_list = ['pie','mash',7,53,6,'eel',2]
>>> b_list = ['mash',6,6.5,'eel','pies']
```

Give a sequence of expressions using only `a_list`, `b_list` and Python operators and methods (and so do not use additional integers, numbers, strings and so on, apart from as indices) that transforms `a_list` into `b_list`.

9. Type:

```
>>> help(list)
```

Write a Python expression, using list methods and operators, to give each of the following, with reference to a list `a_list` (use `a_list` from 8 above to illustrate your expressions):

(a) the index of the 'middle' element of `a_list`, where the middle element of a list of length $2n$ or length $2n-1$ is the nth (illustrate your expression on an even-length list too)

(b) the middle element of `a_list`

(c) the list `a_list` sorted into ascending order (you should assume that all items of `a_list` are of the same type; otherwise, sorting doesn't work)

(d) the list `a_list` with the first element removed and placed after the final element.

10. Type:

```
>>> a_list = ['pie','mash',7,4.3,'eel']
>>> a_string = 'pass'
```

Write Boolean expressions to check whether:

(a) the sum of the third and fourth entries of `a_list` is an integer

(b) the length of `a_list` is more that the length of `a_string`

(c) the second character of `a_string` appears as a character of some item of `a_list`

(d) every item of `a_list` is of type 'string'

(e) `a_string` appears as an item in the list `a_list`

(f) the length of the concatenation of all items of `a_list`, as strings, is less than 20.

11. Type:

```
>>> import turtle
>>> help(turtle)
```

Use the methods of the `turtle` module to draw a football pitch complete with markings - see:

`http://en.wikipedia.org/wiki/Association_football_pitch`

Here are some useful methods with reference to variables `s` and `t`:

- `s = turtle.Screen()` - opens a Turtle screen called `s`
- `s.bye()` - closes the open screen `s`
- `t = turtle.Turtle()` - `t` denotes a turtle object
- `t.forward(distance)` - move the turtle `t` forward the distance `distance` (in pixels)
- `t.left(angle)` - turn the turtle `t` anti-clockwise by `angle` degrees
- `t.right(angle)` - turn the turtle `t` clockwise by `angle` degrees
- `t.undo()` - undo the previous move of turtle `t`
- `t.goto(x,y)` - move turtle `t` to coordinates (x, y) (and draw line if the pen is down)
- `t.setheading(angle)` - set orientation of turtle to `angle`, given in degrees with 0 east, 90 north, etc.
- `t.circle(radius)` - draw a circle of radius `radius` whose centre is `radius` to the left of turtle `t`
- `t.circle(radius,angle)` - draw the arc of a circle of radius `radius`, whose centre is `radius` to the left of turtle `t`, where this arc is subtended by an angle of `angle`
- `t.penup()` - lift the pen of turtle `t` up
- `t.pendown()` - put the pen of turtle `t` down