EEN118 LAB FOUR

The purpose of this lab is to give you more practice with nice clean structured designs. Make absolutely sure your programs are very clear and obviously correct even to a reader who doesn't know what your design plan was. Grading will be on this basis.

The first two parts should be very fast. Remember the `%` operator. If you find that parts 1 or 2 are taking more than just a few minutes to work out, ask the Lab Guy for a hint. You have probably just not realised what an easy thing to do it really is.

1. *What Time Is It?*

The library includes a function called `integer_time()`. It takes no parameters, and returns as its name suggests, the current time of day represented as an integer. The time is always returned as a six-digit `int`, the first two digits are the hour (0-23), the next two are the minute (0-59) and the last two are the seconds (0-59). At exactly quarter past five in the afternoon (17:15 hrs), `integer_time()` would return 171500.

There is also a function called `integer_date()`, which works in a similar way. It always returns an eight-digit int containing the four digit year, then the two digit month, then the two digit day. On 13th February 1997, `integer_date()` would have returned 19970213.

Write a simple program that uses these two functions to determine the current time and date. The time and date should be split into five separate values: year, month, day, hour, and minute. After a little calculation, your program should contain these five lines, or something very much like them:

```
print("  year: "); print(year);   newline();
print(" month: "); print(month);  newline();
print("   day: "); print(day);    newline();
print("  hour: "); print(hour);   newline();
print("minute: "); print(minute); newline();
```

although this requirement is just to emphasise the fact that you must split the time and date properly into its five components. If run at quarter past five on the afternoon of 13th February 1997, it should produce output just like this:

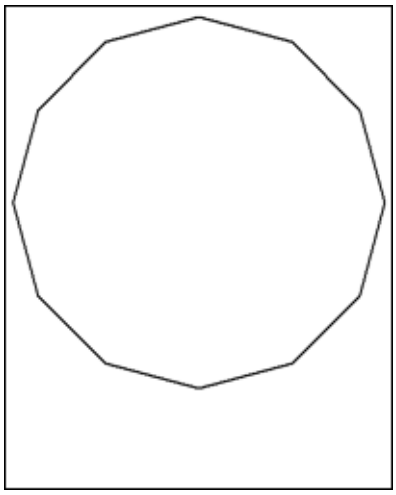
```
  year: 1997
 month: 2
   day: 13
  hour: 17
minute: 15
```

2. *Remove the European Influence.*

Who uses the continental clock? Everyone knows hours are supposed to range from 1 to 12, not zero to 23. Modify your program so that the hour is reduced to the proper 1 to 12 range, but make it also note whether the time is "`a.m.`" or "`p.m.`".

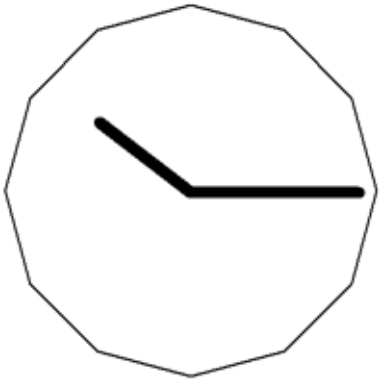
3. *A Clock Face.*

Create a window slightly taller than it is wide. Draw a regular dodecagon ("twelve-a-gon") in the top portion of it. This will become a clock face. The reason for using a twelveagon instead of a normal circle is that it lets you see where the hours are without having to add any special marks or draw on the digits. The result should be something like this:



4. *A Clock with Hands.*

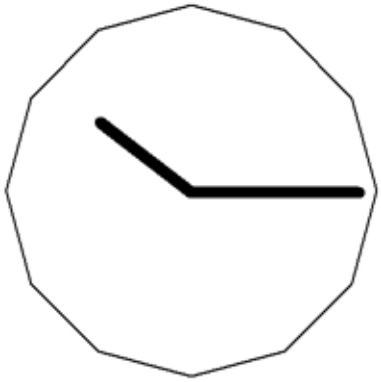
Using your solution to part 2 to determine what the time is, draw on a minute and hour hand indicating the correct current time of day. If you can't immediately calculate exactly where the centre of your clock-face is, a little trial-and-error won't do any harm: make an approximation to where you think the centre is, and draw the hands on that basis; if the result doesn't look right, modify your guess a little. If the program is run at 10:15 (a.m. or p.m.), you should see something like this:



A hint for correctness: think carefully about where the hands should be pointing at (for example) one minute before two o'clock. The time then is 1:59, but where should the hour hand be pointing?

5. *Morning or Afternoon.*

   Use the library function `WriteText` (in conjunction with `MoveTo`) to make an A.M. or P.M. indication appear in the space directly beneath the clock.



6. Your Own Design.

   Animate the clock, and use the area under the clock to display the time and date in a human-friendly form, something like as shown in the diagram below. Perhaps you could even put the numbers 1 to 12 around the edge of the clock face. Make it look as a good clock should. You might find it more interesting to make the date come out in proper words, instead of 1997-2-13 writing something like 13th February 1997.

   Note that the `WriteText` function only accepts strings as parameters. There is a conversion function in the library that will create a string that represents any `int` value. For example, `string_from_int(1997)` returns the string `"1997"`. It would be really nice to see "`13th February 1997`" instead of "`1997-2-13`".

How do you animate the clock? There are no library functions for moving a line once it has been drawn, but *re*moving a line is a very easy thing that you can work out. In fact, removing the whole clock face is even easier.

Of course, your program will have to be in a loop so that it can keep redrawing the clock to show the right time. There is one special thing you need to know to make it work properly. The first time that the library functions find out the time for a program, they record the answer, so they don't have to find out ever again. For programs that only run for a second or two, this works out fine, but for a clock-drawing program it is a disaster: you'll just see the same time being displayed for ever. There is a special library function called `look_at_the_clock()`. Its job is to ensure that the next time `integer_time` is used, it will actually get the current time, and not use the same old recorded time as before.

Also, you will not want to have to stare at your program for a couple of minutes just to be sure that the hands really are moving. Add a second hand, and everyone will have an easier job.