CSC1010H Tutorial 3 – Python Selection and Iteration

Introduction

Learning outcomes

Skills

- Be able to construct and reason about Python solutions to programming problems involving selection and/or iteration.
  Including,
  - Constructing and reasoning about compound Boolean expressions involving the use of python comparison operators.
  - Constructing and reasoning about data selection expressions that use of string slicing and casting.

Knowledge

- The forms of Python if, if-else and if-elif statements.
- Python comparison operators, the Boolean operators 'and', 'not' and 'or', and operator precedence.
- The form of the Python for loop, and the use of break and continue statements.
- The role of the range function in for loops.
- The distinction between definite and indefinite loop structures.
- Forms and possible uses of string slice and casting operations.

Exercise One [25 marks]

This exercise concerns decoding South African identity (ID) numbers.

A South African ID number is a 13-digit number with the following format (courtesy Western Cape Government):

```
YYMMDDSSSSCAZ
```

- The first six digits are the person's date of birth, with 2 digits for year, then 2 digits for month, then 2 digits for day e.g. 6th June 2017 is represented as 170606.
- The next four digits identify the person's gender. Females are assigned numbers in the range 0000-4999. Males are assigned numbers in the range 5000-9999.
- The next digit indicates whether the person is South African citizen, or a permanent resident. A 0 denotes a South African citizen. A 1 denotes a permanent resident.
- The last digit is a checksum digit that is used to check that the number sequence is accurate.
- *The second to last digit is not used.*

Part A *[10 marks]*

Write a program called *simpleparser.py* that asks the user to input their ID number, and then decodes and prints out their date of birth, gender, and citizenship status.

Sample I/O:

```
Please enter your ID number:
9202204720082
```

CONTINUED

```
Your date of birth is 20/02/92.
You are female.
You are a South African citizen.
```

Sample I/O:

```
Please enter your ID number:
1703295010174
Your date of birth is 29/03/17.
You are male.
You are a permanent resident.
```

The Luhn algorithm

The check digit at the end of an ID number is used to detect accidental mistakes, such a through misreading or mistyping. Given an ID, the validity (correctness) can be determined by recalculating the check digit and comparing it to that which is given.

A check digit is calculated by applying the Luhn algorithm to the first 12 digits:

$$d_{11}d_{10}d_9d_8d_7d_6d_5d_4d_3d_2d_1d_0d_c$$

Essentially, the algorithm requires calculating the sum of the digits i.e.

$$sum = d_{11}+d_{10}+d_9+d_8+d_7+d_6+d_5+d_4+d_3+d_2+d_1+d_0$$

However, alternate digits are not added directly. For each of these, multiply by 2, obtain the modulus of 9, and add this to the total. (i.e for each d in $d_{10,}d_{8,}d_6d_4d_2d_0$, obtain $(d \times 2)\ modulus\ 9$).

The full summation is:

$$sum = d_{11+}(d_{10} \times 2)\ modulus\ 9 + d_9 + \cdots + d_1 + (d_0 \times 2)\ modulus\ 9)$$

Calculate the expected check digit by obtaining $sum\ modulus\ 10$.

An ID number is valid if:

$$10 - (sum\ modulus\ 10) = d_c$$

Part B *[15 marks]*

Write a program called *advancedparser.py* that asks the user to input their ID number, and then does two things:

(i) using the check digit, it determines whether the number is valid;
(ii) (ii) if the number is valid then it decodes and prints out the user's date of birth, gender, citizenship status.

Sample I/O:

```
Please enter your ID number:
9202204720082
Invalid ID number.
```

CONTINUED

Sample I/O:

```
Please enter your ID number:
1703295010174
Your date of birth is 29/03/17.
You are male.
You are a permanent resident.
```

Exercise Two [25 marks]

In South Africa, the department of transport issues different driving licenses depending on the category (or categories) of vehicle that the recipient wishes to drive (or learn to drive).

How is a vehicle classified? It depends on the Gross Vehicle Mass (GVM), and on whether the vehicle is 'articulated', or whether a trailer is being towed.

The GVM dictates the base class to which the vehicle belongs: B, C1, or C.

GVM≤3500	Class B
3500<GVM≤16000	Class C1
16000<GVM	Class C

If, however, the vehicle is articulated, or is towing a trailer with a GVM greater than 750kg, then it belongs in an extended, 'E', version of the class: EB, EC1, or EC. (According to Wikipedia, an articulated vehicle is one "which has a permanent or semi-permanent pivoting joint in its construction".)

The classification scheme does not allow for an articulated vehicle towing a trailer. It's an 'either-or' situation.

Write a Python program called *vehicleclassifier.py* that can calculate the class to which a vehicle belongs.

Here are three examples of what should happen when the program runs:

```
What is the gross vehicle mass (in kg)?
2500
Does the vehicle have a trailer?
y
What is the gross vehicle mass of the trailer (in kg)?
400
This vehicle is class B.
```

```
What is the gross vehicle mass (in kg)?
4000
Does the vehicle have a trailer?
n
Is the vehicle articulated?
y
This vehicle is class EC1.
```

```
What is the gross vehicle mass (in kg)?
3000
Does the vehicle have a trailer?
y
```

CONTINUED

```
What is the gross vehicle mass of the trailer (in kg)?
1000
This vehicle is class EB.
```

RECOMMENDATION: Develop your program in stages (as was done in exercise one). Start with a version that can determine the base class only:

```
What is the gross vehicle mass (in kg)?
2500
This vehicle is class B.
```

Extend your program with questions and tests to see if the vehicle belongs in an extended class. (HINT: Use string concatenation to add 'E' to the front of the base classification if it does.)

Exercise Three [25 marks]

This exercise concerns converting measurements in feet and inches (imperial system) to metres (metric system).

Part A *[15 marks]*

Write a program called *feetmetres.py* that displays a conversion table for measurements in feet to measurements in metres. The program should ask the user to enter the range of values that the table will hold. Here is an example of what should be output when the program runs:

```
Enter the minimum number of feet (not less than 0):
5
Enter the maximum number of feet (not more than 99):
10
   5' |   1.52m
   6' |   1.83m
   7' |   2.13m
   8' |   2.44m
   9' |   2.74m
  10' |   3.05m
```

RECOMMENDATION: Develop your solution in parts. Try, for example, printing just the feet part before you start working on the conversion.

```
Enter the minimum number of feet (not less than 0):
5
Enter the maximum number of feet (not more than 99):
10
   5' |
   6' |
   7' |
   8' |
   9' |
  10' |
```

Leave out the feet symbol (" ' ") initially, if it makes things easier.

CONTINUED

Part B *[10 marks]*

Write a program called inches*metres.py* that displays a conversion table for inches to metres. The program should ask the user to enter the range of values that the table will hold. Here is an example of what should be output when the program runs:

```
Enter the minimum number of inches (not less than 0):
1
Enter the maximum number of inches (not greater than 11):
6
Inches:    1    2    3    4    5    6
Metres: 0.03 0.05 0.08 0.10 0.13 0.15
```

Metre values are given to two decimal places.

RECOMMENDATION: Develop your solution in parts. Try, for example, printing just the inches line, then try printing the metres line. In each case you'll need a loop that runs through the given range of values.

To discover how to keep all your numbers on one line, try entering and running the following python programs:

(a)

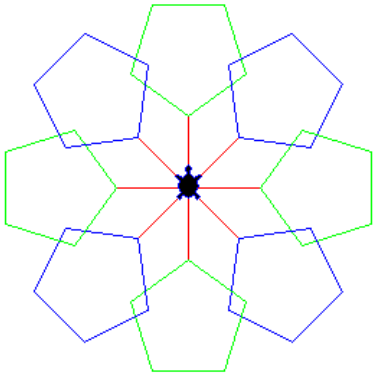
```
print('Hello')
print('World')
```

(b)

```
print('Hello', end='')
print('World')
```

Exercise Four [25 marks]

Consider the following flower pattern:



It consists of 8 petals (polygons) on red stems (spokes). The petal colour alternates between green and blue.

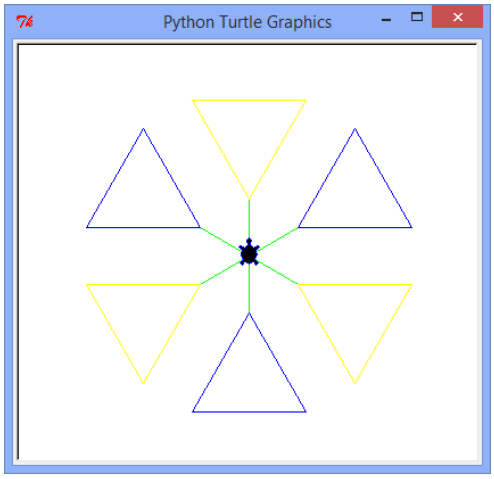
Write a Python Turtle program called *flowerpattern.py* which allows the user to draw patterns of this sort. The program will ask the user to enter the number of petals, the length of stem, the sort of petal (triangle, square, or pentagon), the length of petal side, and the colours.

Here is a sample program execution:

```
Enter the number of petals (must be an even number): 6
Enter the length of stem: 50
Enter the petal type (triangle, square, or pentagon): triangle
```

CONTINUED

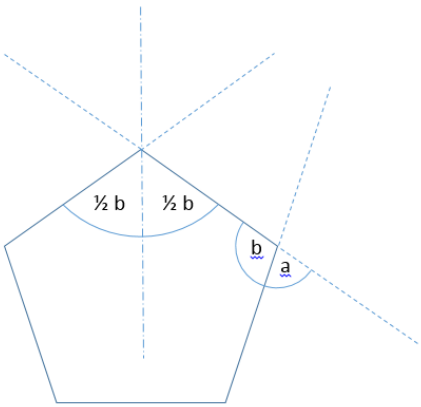
```
Enter the length of side: 100
Enter the stem colour: green
Enter the first petal colour: blue
Enter the second petal colour: yellow
```



RECOMMENDATION: As with other exercises, try breaking the problem down into simpler tasks and then building and combining solutions. For example,

- Write a program that obtains the required inputs (and translates petal type to the required number of polygon sides).
- Extend the program to draw the stems/spokes.
- Write a program that asks the user to enter 3, 4, or 5, and draws a polygon with that many sides.

HINT: How do you centre a petal on its stalk? The turtle will turn a certain amount clockwise or a certain amount anti clockwise before drawing the polygon (depending on whether it is drawn anti clockwise or clockwise):



Facts: (i) *a=360/(number of sides)* and (ii) *a+b=180.*

Marking and Submission

Submit the *simpleparser.py*, *advancedparser.py*, *vehicleclassifier.py*, feetmetres.py, inchesmetres.py, and *flowerpattern.py* files contained within a single .ZIP folder to the automatic marker. The zipped folder should have the following naming convention:

CONTINUED

yourstudentnumber.zip

Marking Guide

Exercise One – ID numbers	25
Exercise Two - Vehicle Classification	25
Exercise Three - Metric to Imperial Conversion	25
Exercise Four - Python Turtle Polygons	25
Total	**100**

Note that question 5, *FlowerPattern.py*, will be marked manually.

END