CM1103 Lab worksheet week 5

Operators & math functions *[Think Python Version 2 - Sections 1.4–1.5, 2.3–2.5, 3.2]*

1. Assume that we execute the following assignment statements:

```
width = 17
height = 12.0
delimiter = '.'
```

For each of the following expressions, write the value of the expression and the type (of the value of the expression).

(a) `width` 17 - int

(b) `width / 2` 8.5 - float

(c) `width // 2` 8. - int

(d) `height // 7` 1.0 - float

(e) `1 + 2 * 5` 11 - int

(f) `delimiter * 5` ..... - str

Use the Python interpreter and the type function to check your answers.

2. (a) Use the interactive shell to work out how many seconds in total are in 3 hours, 45 minutes and 2 seconds. 13502 seconds

(b) Replace each occurrence of `???` in the following so that it converts the number of seconds (6149) to hours, minutes and seconds, and outputs these values.

```
>>> totalSeconds = 6149
>>> seconds = totalSeconds ??? 60     %
>>> totalMinutes = totalSeconds ??? 60        //
>>> minutes = ???   totalMinutes % 60
>>> hours = ???     totalMinutes // 60
>>> print(hours, minutes, seconds)
1 42 29
```

3. What happens if you pass the `sqrt` function that we implemented in the lectures a *negative* number as a parameter? It triggers a runtime error, math domain error to be exact

4. Typing

```
>>> import math
```

in an interactive Python shell will let you use functions from the built-in `math` module. Type:

```
>>> help(math)
```

to see details of the functions this makes available. Using this module, calculate:

(a) The sine of 15 math.sin(15)

(b) $63^{2.5}$ math.pow(63, 2.5)

(c) The square root of 2498 math.sqrt(2498)

(d) π math.pi

5. Attempt exercise 2.2 from *Think Python Version 2*.

1. (4/3) * math.pi * math.pow(5,3) 2. (24.95*0.4+3) + (24.95*0.4+0.75)*59

3.
```
totalSeconds = (6*60*60) + (52*60)
totalSeconds += (8*60)  + 15
totalSeconds += ((7*60)+12)*3
totalSeconds += (8*60 + 15)
seconds = totalSeconds % 60
totalMinutes = totalSeconds // 60
minutes = totalMinutes % 60
hours = totalMinutes // 60
print(hours, minutes, seconds)
```

7:30:6 AM

Defining functions *[Think Python Version 2 - Sections 3.4, 3.7, 3.10]*

6. Write a function that takes a number of seconds as an argument and displays the number of hours, minutes and seconds.

7. Write a function that takes a number of hours, minutes and seconds as arguments and **returns** the total number of seconds. Add a suitable doctest to this function, and check that it passes.

8. Write functions that perform the calculations specified in parts 1 and 2 of exercise 2.2 from *Think Python Version 2* for any input values, giving careful thought to your choice of parameters.

Indexing and slicing lists *[Think Python Version 2 - Sections 10.2, 10.5]*

Let `a` be a list. The *indexing operator* `a[n]` gives access to the n^{th} element of the list. The *slicing operator* `a[i:j]` gives access to a subsequence of the list, and can be given an optional `step` allowing the subsequence to skip elements.

Define a list containing the integers 0, 1, 2, …, 9 by `a = list(range(10))`.

9. What is output by the following:

   (a) `a[2]` 2

   (b) `a[10]` index out fo range error

   (c) `a[-3]` 7

   (d) `a[0:3]` [0,1,2]

   (e) `a[:3]` [0,1,2]

   (f) `a[4:]` [4,5,6,7,8,9]

   (g) `a[:]` [0,1,2,3,4,5,6,7,8,9]

   (h) `a[::2]` [0, 2, 4, 6, 8]

   (i) `a[5::-1]` [5, 4, 3, 2, 1, 0]

   (j) `a[::2][3]` 6

10. Replace each $*$ with a single character in the following to give the desired output:

    (a) `a[*]` to give 4 4

    (b) `a[-*]` to give 4 6

(c) `a[*:*]` to give `[0, 1]` 0:2

(d) `a[:*]` to give `[0, 1, 2]` 3

(e) `a[-*:]` to give `[8, 9]` 2

(f) `a[::*]` to give `[0, 3, 6, 9]` 3

(g) `a[::**]` to give `[9, 6, 3, 0]` -3

11. Provide code containing one or more slice which gives:

    (a) All even numbers in `a` in ascending order. a[::2]

    (b) The reverse of `a` a[[::-1]

    (c) All odd numbers in `a` in descending order. a[::-2]

    (d) The two highest odd numbers in `a` in descending order. a[::-2][:2]

12. What do the following do to `a` (reinitialise `a` before each) – try to work these out **before** running the code to check your answer:

    (a) `a[0] = 10` first item in the list is now 10

    (b) `a[2:4] = ["a", "b"]` changes items in the list at index 2 and 3 to "a" and "b" respectively

    (c) `a[2:4] = ["a", "b", "c", "d"]` changes the elements at index 2 and 3 and adds 2 new items

    (d) `a[2:4:2] = ["a", "b"]` triggers an error, trying to assign 2 items a sequence of 1

    (e) `a[2:6:2] = ["a", "b"]` Changes item at index 2 and 4

    (f) `del a[0:2]` deletes items at index 0 and 1

    (g) `del a[::2]` deletes every 2nd element

    (h) `a[1::2] = a[::-2]` swaps the items(every other element from index 1) and items(every other element from the last index)

Built in functions

Python has a number of useful built in functions:

`https://docs.python.org/3/library/functions.html`

13. Define `a = list(range(9,-1,-1))`. Use an appropriate built in function and slices of `a` to find (assume you don't know the contents of `a`):

    (a) The number of elements in `a`.

    (b) The sum of all elements in `a`.

    (c) The sum of the first 5 elements in `a`.

    (d) The smallest value in `a`.

    (e) The maximum of the elements in `a` whose index is a multiple of 3.

    (f) A copy of `a` in ascending order.

14. The `statistics` module also contains some useful functions for working with lists. Import the module (as in question 4) and calculate the mean, median and mode[1] of the list `[9, 3, 5, 4, 10, 4, 3, 2, 4]`.

[1] See `http://www.bbc.co.uk/bitesize/ks2/maths/data/mode_median_mean_range/read/1/` if you need to revise these terms.

String manipulation *[Think Python Version 2 - Sections 8.1–8.3]*

15. (a) What does the following code do?

```
>>> for char in "1bc4":
...     print(char)
```

prints each character in a new line
string

(b) What type does the variable `char` have in the above loop? (Hint: use the `type()` function.)

(c) Use the `dir()` and `help()` functions (with suitable arguments) to find a function that can be used to determine whether a string contains only alphabetic characters (i.e. only letters – no numbers or whitespace).

(d) Modify the loop above so that it prints out whether each character is a letter. I.e. it should produce the output:

```
False
True
True
False
```

(e) Modify the loop above so that it considers each character in turn. If it is a letter, it should print it **converted to upper case**. If it is not a letter, it should print `not a letter`. I.e. it should produce the output:

```
not a letter
B
C
not a letter
```

Formative assessment

The question in this section will help to prepare you for the coursework that will be set in week 9.

Download the file `Week5Template.py`, and complete the function as indicated below. Try to ensure your code passes all of the doctests.

16. The *Doomsday algorithm* gives an easy way to compute the day of the week for a given date, and is particularly suitable for mental calculation[2].

    The method represents each day of the week as a number, with `0`=Sunday, `1`=Monday, up to `6`=Saturday, and calculates the *doomsday* for a given year as the day of the week that some memorable dates (4/4, 6/6, 8/8, 10/10, 12/12, and the last day of February) fall on. For example, in 2012 each of these dates falls on a Wednesday. All other dates can then be easily worked out relative to the doomsday.

    The algorithm to work out the doomsday for a given year is as follows:

    ```
    Set x to be 5 if the year is in the century 1800–1899;
        3 if it falls in 1900–1999; 2 if it falls in 2000–2099; and 0 if it
        falls in 2100–2199.
    Set w to be the year of the century for the date (e.g. 10 for the year 2010,
        99 for 1899)
    Divide w by 12 and call the quotient a and the remainder b
    Divide b by 4 and call the quotient c
    Divide the sum of a, b and c by 7 and call the remainder d
    Count forward d days from x to get the year's doomsday (be
        careful if this wraps round into a new week – you should end up with an
        answer from 0,1,...,6)
    ```

    Complete the function `doomsday` so that it calculates and returns the doomsday for a given year using this algorithm. If the function is called with a year outside of the specified range specified in the algorithm (i.e. 1800 – 2199), it should return `-1`.

Advanced

In addition to the optional further work from the lecture slides:

17. Use `help` to find out what the functions `chr` and `ord` do. Write some code that outputs all of the characters represented by numbers 0 up to 255. What does printing `chr(7)` do?

[2] See http://en.wikipedia.org/wiki/Doomsday_rule for more detail.

Challenge: ***Can't see the wood for the trees***

Download the file `message.py` from Learning Central. Start Python and change to the directory containing `message.py`. Type in the following:

```
from message import *
```

You now have access to a long string named `s` containing a question. Email AllenSM@cardiff.ac.uk with your answer and any code you wrote to solve the problem, with the subject line "CM1103 week 5 Challenge" by midnight on Wednesday in Week 6 – a winner will be picked at random from the correct answers **but** you'll get one extra entry for particularly noteworthy or interesting code in your answer.

[Hint: What does `s.find("W")` output?]