MASSEY UNIVERSITY

159.171 – Computational Thinking and Software Development

Assignment 6 – Semester 1, 2013

Lay your programs out so they look good to you – *exactly mimicking any sample output is NOT required.*

Submit a document (DOC/DOCX/ODT) containing both

- screenshots of your programs running.
- a listing of each program formatted using a 10pt font, or whatever is require to avoid the lines wrapping. **Submissions without complete listings will not be marked**.

Program 1: A Weekly Pay Table - 6 marks

Write a Python program that calculates the weekly gross pay for pay rates from \$12 to \$20 per hour, working from 5 to 20 hours per week, and displays the results. An example of the layout formatting (using different numbers) is shown to the right.

IMPORTANT

You MUST use two for-loops, one inside the other.

Solutions creating the results by adding lines like:

```
printNow ('12 hrs/wk:  60   72   84   96 ....');
printNow ('13 hrs/wk:  65   78   91  104 ....');
```

will be awarded zero marks.

	Pay Rate: Dollars per Hour					
	5	6	7	8	9	10
10 hrs/wk:	50	60	70	80	90	100
11 hrs/wk:	55	66	77	88	99	110
12 hrs/wk:	60	72	84	96	108	120
13 hrs/wk:	65	78	91	104	117	130
14 hrs/wk:	70	84	98	112	126	140
15 hrs/wk:	75	90	105	120	135	150
16 hrs/wk:	80	96	112	128	144	160
17 hrs/wk:	85	102	119	136	153	170
18 hrs/wk:	90	108	126	144	162	180
19 hrs/wk:	95	114	133	152	171	190
20 hrs/wk:	100	120	140	160	180	200
21 hrs/wk:	105	126	147	168	189	210
22 hrs/wk:	110	132	154	176	198	220
23 hrs/wk:	115	138	161	184	207	230
24 hrs/wk:	120	144	168	192	216	240
25 hrs/wk:	125	150	175	200	225	250

Sample output format. Your program displays a similar table but with different numbers.

Hints:

- line two of the header (the 5 6 7 …) can be created using a fixed string (for the above example):

  ```
  '          5    6    7    8    9    10'
  ```

  or another *for-loop* - whichever you prefer.

- tabs between the items of output can help in getting even spacing.

- get one *for-loop* (the one that creates a single row) working correctly, then wrap it in another *for-loop*, that creates new rows.
  e.g. - you could have a variable called *hoursPerWeek* and set it to 5.
  - use a single *for-loop* to create the first row of the table.
  - once this is going, take the *for-loop* statement and surround it with another *for-loop* that changes *hoursPerWeek*.

Program 2: A Romance Compatibility Calculator – 4 marks

There is a theory (which I've just thought up based on absolutely no evidence) that it's possible to calculate the romantic compatibility between two people using just their names. The process for doing this is:

- derive a person's score by treating each letter of their name as a number and adding them all up.
- find the remainder when dividing this number by 100 (e.g. 537 gives 37, 401 gives 1 …)
- find the difference between the two scores, returning the absolute value (e.g. convert -37 to 37)
- display this value as a percentage compatibility value (based on the dubious supposition that the larger the difference in scores, the more compatible you are). The value isn't a percentage but it looks better when displayed that way.

Write a program to calculate the compatibility value for two names.

- if the names are different and neither is blank, display their compatibility.
- if the names are the same, or either is empty, display an *appropriate* error message informing the user what's wrong.

Picking out the characters

Method #1: You can pick out individual characters from a string using subscripts (e.g. if the string *myname* contains 'Fred', *myname[0]* is the 'F', and the final character (the 'd') is in *myname[3]*.

Method #2: this is simpler than method #1. It uses the *for – in* construct*:*

```
for ch in s:                # where s is any string and ch is the picked-out character
    do something with the character ch
```

To find the integer value of a character use the *ord()* function:

```
intValue = ord(myname[2])   OR
intValue = ord(s)
```

Once you've can find the value of each character, you add them up. The accumulator pattern is useful here.

Important: To ensure that you don't have any spaces before or after the name, use the strip*()* method: **`e.g. myname = myname.strip()`** as trailing spaces won't be visible but will affect the numeric total.

Remainders: You can use the **%** operator to find remainder of a division: e.g. *14 % 5* gives *2*

Test data:

Here are some test values to see whether your program is working correctly. Only the names and final compatibility value need to be displayed, but you can also display the intermediate values if you wish.

Names	Sum of the letters	Remainders	Difference	Compatibility
Igor & Belinda	401 & 687	1 & 87	-86	86%
bill & mary	419 & 441	19 & 41	-22	22%
Bill & Mary	387 & 409	87 & 9	78	78%
Frodo & Mary	506 & 409	6 & 9	-3	3%

It's obvious that the method is suspect. *Bill* and *Mary* are given a much higher compatibility score than *bill* and *mary,* which doesn't seem to translate to what we know about the real world …

On the next page, there's a table of character values that you can use to "desk-check" your assignment (manually performing the same steps as your program to ensure it's correct).

Testing – a fragment of code you can use if you wish

Here's a piece of code I used when testing my solution. It feeds the known test values to a function that does the work.

```
for s in [ 'Igor+Belinda', 'bill+mary', 'Bill+Mary', 'Frodo+Mary']:
    name1, name2 = s.split('+')
    romanceCompatibility(name1, name2)
```

Each time around the loop:

1. the string *s* is set to the next string in the list (e.g. 'Igor+Belinda').
2. The ***.split()*** method returns a list given by splitting the string whenever a **'+'** occurs, so the result of *s.split('+')* is a list of two items (the names).
3. the first item in the list is put into *name1* and the second into *name2*
4. my function was called to calculate and display the compatibility.\

While the lines above are handy for testing, make sure you comply with the specification above that require you to prompt for the names and do some checking before calculating their compatibility.

Appendix for Program 2 – a section of the ASCII Character Table

The numeric values of the characters are based on values assigned to letters in the international character code called *ASCII* (American Standard Code for Information Interchange). We'll be dealing with this in the *Data Representation* section of the paper, but if you'd like to desk-check the values from your program (you can get it to display the totals), here are the values for the alphabetic characters:

You do NOT need to type this table or the character values into your program:

- characters in a Python program will already have the correct values, you just have to get your program to added them up for each person and then subtract one from the other.

- the use of this table is optional. It's here so you can manually check whether your program is doing what you think it is.

e.g.

"Bill"	will give:	66 + 105 +108 + 108	➔ 387
"bill"	will give:	98 + 105 +108 + 108	➔ 419
"BILL"	will give:	66 + 73 + 76 + 76	➔ 291

A = 65	a = 97
B = 66	b = 98
C = 67	c = 99
D = 68	d = 100
E = 69	e = 101
F = 70	f = 102
G = 71	g = 103
H = 72	h = 104
I = 73	i = 105
J = 74	j = 106
K = 75	k = 107
L = 76	l = 108
M = 77	m = 109
N = 78	n = 110
O = 79	o = 111
P = 80	p = 112
Q = 81	q = 113
R = 82	r = 114
S = 83	s = 115
T = 84	t = 116

U = 85	u = 117
V = 86	v = 118
W = 87	w = 119
X = 88	x = 120
Y = 89	y = 121
Z = 90	z = 122
a space = 32	the hypen (-) = 45

