CS150 - Test Project 1
Due: Wednesday Oct. 24, before the beginning of class

A test project is an assignment that you complete on your own, without the help of others. It is a form of take-home exam. You may use the textbook, your notes, your previous assignments, the notes and examples on the course web page and the Python library documentation (linked on the course web page), but use of any other source is forbidden. You may *not* discuss these problems with anyone aside from the course instructor. You may only ask the tutors for help with hardware problems or difficulties in retrieving or submitting your program.

Complete each of the three problems below, each as a separate program, in a separate file. You are encouraged to reuse code from your assignments or our class examples. Partial credit will be awarded, so if you can't solve a whole problem, get as far as you can.

Improving the programs and extra credit

If you do everything that is stated below, the maximum number of points you can get is 60. In order to get the full credit of 63, you must implement some extra features for some of the problems. I've provided some ideas below, but you should feel free to exercise your creativity. In addition, you may receive up to 3 points of additional extra credit (i.e. for a total of 66). You may receive at most 3 points of extra credit for any given program.

For all additions, include in the comments at the top of the file what you've added or you may not get credit.

1 Taxes

> *But in this world nothing can be said to be certain, except death and taxes.*
>
> –Benjamin Franklin

In preparation for entering the real-world, we're going to write a program that roughly estimates your federal taxes for the year. When you run the program:

- The user should be prompted for:
  - their hourly wage

    - the number of hours they work a week

- The program will then print out how much the user makes in a year
- and an estimate or how much they make per week *after* taxes have been taken out

For example, here is a sample transcript from the program running:

```
*****  Salary Calculator  *****
Enter your hourly salary: 25
Enter your hours per week: 45

---------  Results -----------
You make $61750.0 a year
After taxes you will make $890.625 per week
```

You **must** write the following functions:

- `salary_calculator`: Takes two parameters, hours worked in a week and the hourly wage and calculates the weekly wage. The first 40 hours a person works in the week are at the normal wage. Any hours above 40 that a person works are overtime and are billed at 1.5 times their normal wage. For example, if a person works 50 hours at $10/hour then they would make $550 ($400 regular and $150 of overtime).
- `tax_bracket`: Takes a single parameter, a *yearly* income, and returns the tax bracket as a the fraction of income that would be required to be payed, e.g. 10% would be 0.1. You can find the tax brackets online at:

  `http://www.efile.com/tax-service/tax-calculator/tax-brackets/`

  We will use these as hard cutoffs, for example, if you made $30,000, `tax_bracket` would return 0.15[1]
- `salary_after_taxes`: Takes as input a yearly salary and returns the amount of money a person with that yearly salary would make after taxes have been deducted (based on their tax bracket).

In addition to these functions, you may also include other functions and will need to include additional code to make your program behave as described above.

Possible program extensions: print the dollar figures with the appropriate number of decimal points (e.g. $890.62); calculate the projected salary after some number of years assuming a fixed percent salary increase; allow the user to enter some tax write-offs that would decrease the yearly salary; calculate the *real* taxes based on progressive tax brackets.

[1] In actuality, taxes are progressive, in that the first $8500 you make is taxed at 10%, then the amount from $8500-$34,500 at 15%, etc.

2 Vocabulary Game

Write an interactive program that when you "run" the program it starts playing the word game:

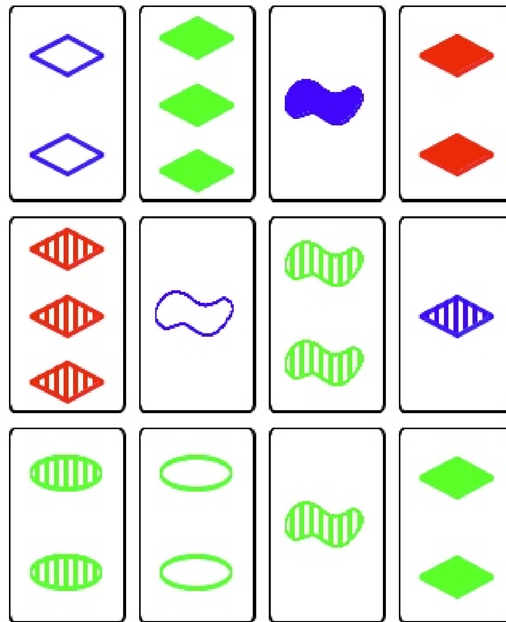
```
Would you like to play a word game? yes
How many seconds do you want to play for? 10
Enter a word starting with 'm': middlebury
Great!
Enter a word starting with 'm': Marmot
Great!
Enter a word starting with 'm': nano
nano doesn't start with 'm'
Enter a word starting with 'm': mascot
Great!
You named 3 words that start with 'm' in 10 seconds
```

- Your program should follow the format above precisely.
- You must use at least 2 functions in your program.
- When you ask the user if they want to play the game, if they say any capitalization variant of "yes" (e.g. "YES", "Yes", "yEs", ...) then the game should continue. If they enter anything else, then the game should just end. (Hint: there's a straightforward way of doing this without having to check all the possibilities.)
- The starting letter should be randomly selected each time the game is played.
- Your program must check if the word starts with that letter, though it does NOT need to check if it's an actual word. If an entered word starts with the right letter, a positive message should be printed. If not, the user should be told that that word does not start with the letter.
- The user should be able to enter words that are lowercase or capitalized and still work properly.

I'm giving you a fair amount of flexibility about how you break this program into different functions. Think about the best way to break this into smaller components and use good style!

Possible program extensions: check if the word is in the dictionary (see the class notes on 10/10 for a list of English words); rather than one point per correct word, assign points based on the length of the word; offer the player an alternative version of the game (in addition to the original) that times how long it takes to list some predetermined number of words; add other constraints to the words entered besides just the first letter constraint

3 Set Game



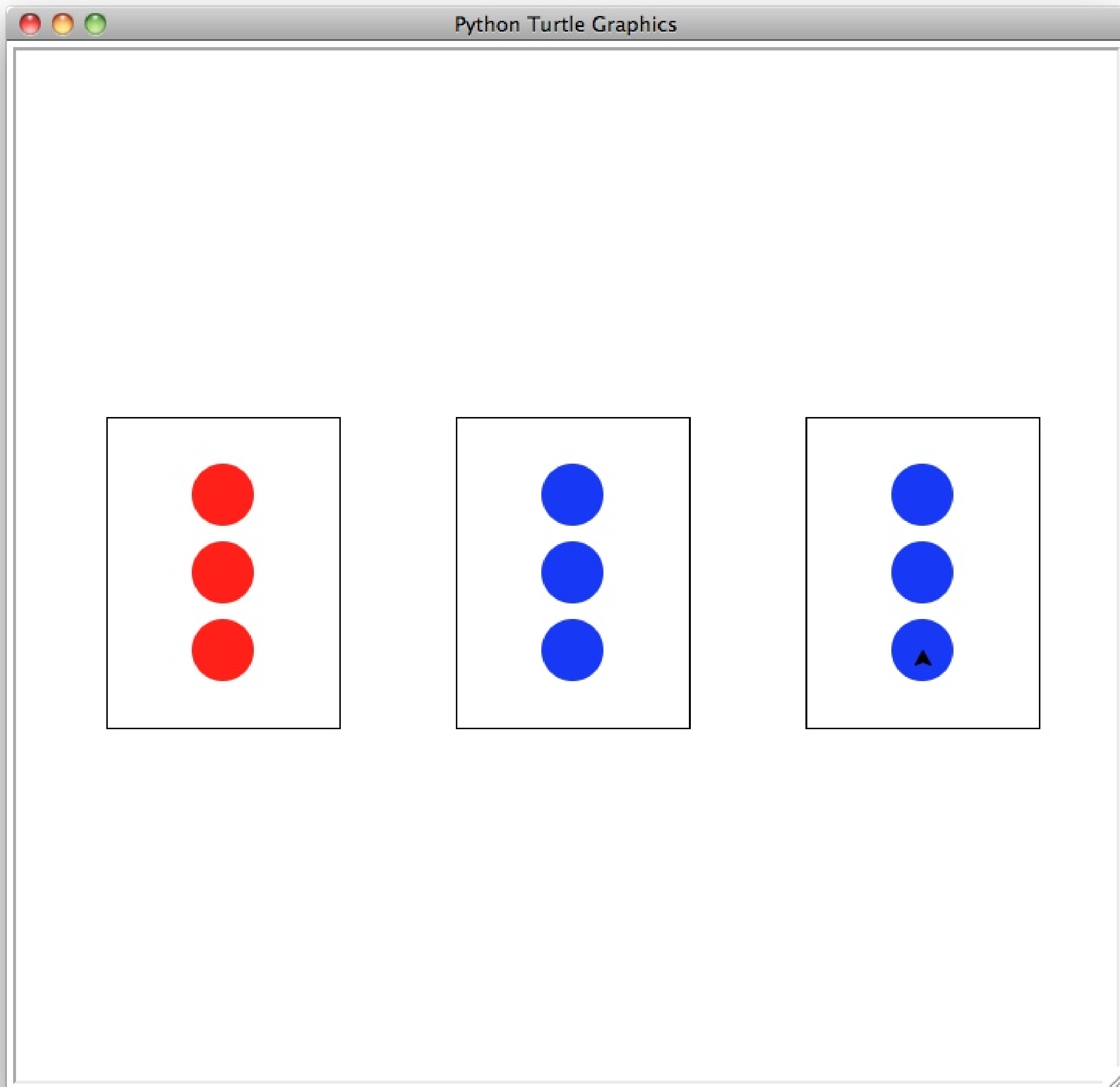
Set is a card game that starts with 12 cards laid out on the table. The set cards have a number of attributes including color, type of shape, number of shapes and the texture of the shape. A "set" is a collection of three cards where each attribute is either the same (e.g. all red) or all different (1, 2, 3 shapes). See

`http://www.setgame.com/set/rules_set.htm`

for the actual rules, examples, etc.

We're not going to implement this full game, but are going to get the ball rolling by writing a program that will generate some Set-like cards randomly using `turtle` graphics. Specifically, you should write a program that generates three cards evenly spaced on the screen. The cards will all have three circles/dots for the shape and each card will have a randomly chosen color picked from: blue, red and green.

For example, here is a sample output from a run of the program:



- You must implement a function called `generate_random_card` which takes two parameters, the x and y location of the top left corner of the card, and draws a card with three circles/dots with a random color.
- You must implement at least one other function to help you draw the three cards.
- You must use at least one `for` loop somewhere in your code.

It may be easier to use the `dot` function, instead of the `circle` function, though you may use either. See the `turtle` documentation for using `dot`.

Possible program extensions: generate 9 (or 12) cards instead of three (if you do this, you must use one or more loops to generate the cards); add more card variations, such as shape, texture and number of circles; add a caption to the picture (see the `write` function)

When you're done

You should have three separate files/programs, one for each of the programs above. Make sure that each program is properly commented:

- You should have comments at the very beginning of each file stating your name, course (including section number), test project number and the date.
- Each function should have an appropriate *docstring*
- Other miscellaneous comments to make things clear

Submission

Submit your file digitally online. To do this, you'll need to create a single file as follows:

- Create a folder with your name on it, e.g. I would create a folder and call it "kauchak".
- Put all three of your .py files, one for each of the problems, in the folder.
- "zip" up the folder (i.e. create a .zip file from the folder)
  - **Mac:** Right-click on the folder (i.e. ctrl + click) and select "compress".
  - **Windows:** Right-click on the folder and select "Send to" then "Compressed (zipped) folder"
- Upload this final .zip file digitally as usual. Enter "TP1" for the assignment number field.

Grading

		points
style/comments (per program)		8 * 3
	if/while/for	
	variable naming	
	comments/docstrings	
	formatting	
	parameters	
	additional functions	
	misc	
Taxes		
	`salary_calculator`	3
	`tax_bracket`	3
	`salary_after_taxes`	2
	User input and program output	4
Vocabulary Game		
	play game or not	2
	timing works	2
	random letter	1
	check entered words	2
	proper counting	1
	proper output	2
	used 2 functions	1
	general functionality	1
Set Game		
	card placement	3
	circle placement	3
	random color	2
	use `for` loop	1
	use another function	1
	`generate_random_card`	2
Required extra add-ons		3
extra credit		3
total		62 (+3)