

Informatics Assignment

What to turn in

Turn in a printout of your code exercises, stapled to your answers to the written exercises, on or before January 15th.

Exercise 1.0 – Installing Python

Follow the instructions on installing Python and IDLE on your own computer on the **setup** page of the course website, <https://engr-rudn.github.io/python-novice-inflammation/setup.html>.

Exercise 1.1 – Hello, world!

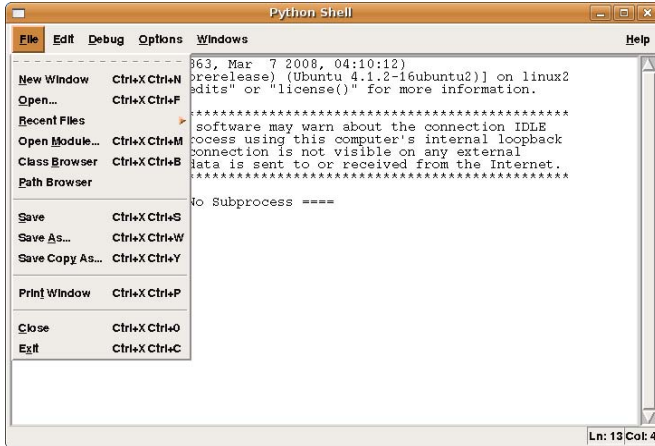
Recall that a program is just a set of instructions for the computer to execute. Let's start with a basic command:

`print(x)`: Prints the value of the expression *x*, followed by a new line.

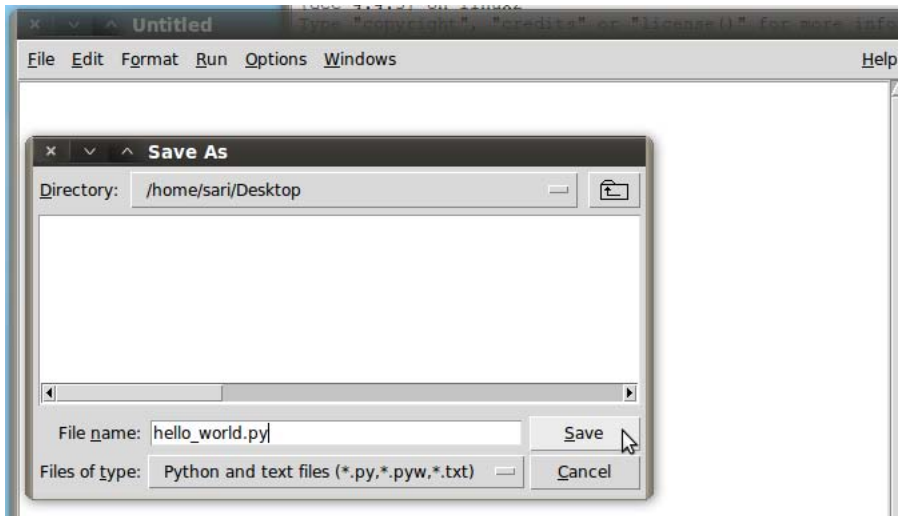
Create a new file called `hello_world.py`. You will use this file to write your very first 'Hello, world!' program, as well as your answers for the rest of the exercises for today. How to create a program file:

You can use Python IDLE or Anaconda to do this assignment. The instructions here use the Python IDLE for demonstration purposes. But still you can use Anaconda and Jupyter Notebook for writing your code.

1. Open a new window by choosing **New Window** from the **File** menu.



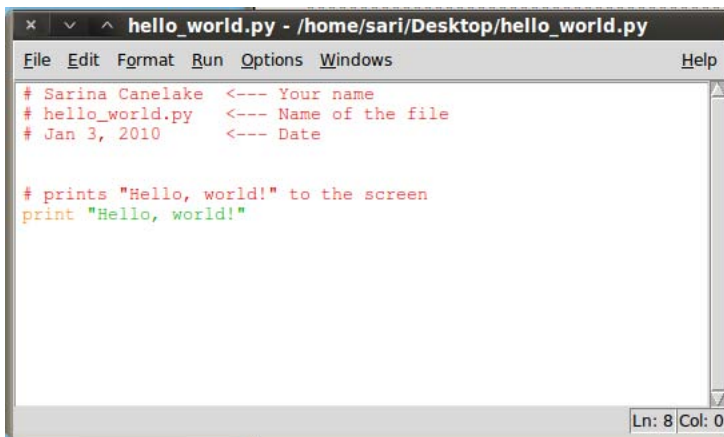
2. Save the file as `hello_world.py`. Do NOT skip the `.py` portion of the file name - otherwise, you will lose out on syntax highlighting!



3. Start every program with a bank of comments, with a comment line for your name, your recitation section, the name of your file, and today's date. Recall that a comment line begins with a `#` (pound) symbol.

You can now write your very own Hello, world! program. This is the first program that most programmers write in a new programming language. In Python, Hello world! is a very simple program to write. Do this now... it should be only be one line!

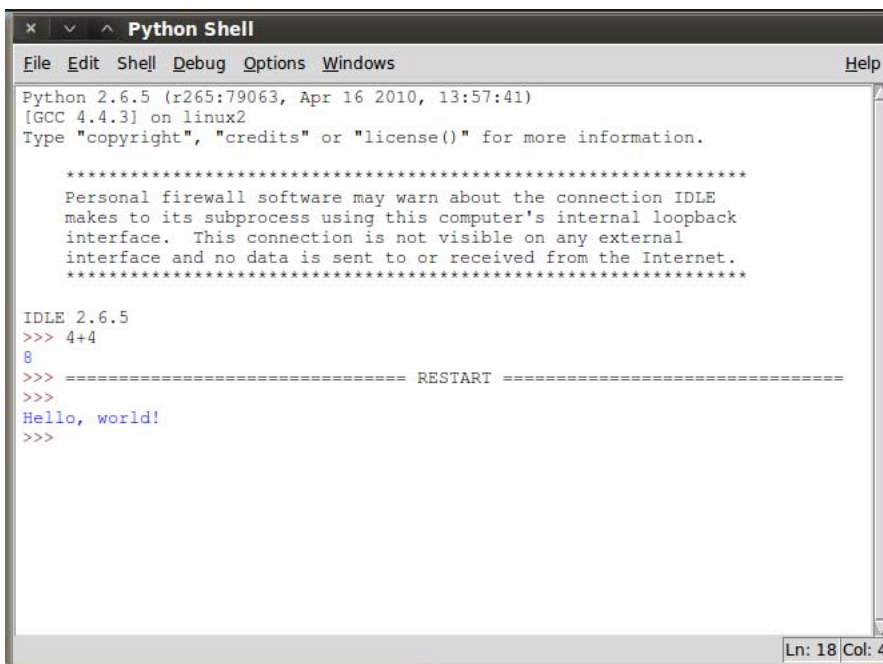
When you are done, save your work and run it. Your code should look similar to this:



```
# Sarina Canelake <--- Your name
# hello_world.py <--- Name of the file
# Jan 3, 2010 <--- Date

# prints "Hello, world!" to the screen
print "Hello, world!"
```

To run your program, chose **Run Module** from the **Run** menu (or just hit F5 on Windows/Linux, or fn-F5 on a Mac). When you run the code, your shell should look similar to this:



```
Python 2.6.5 (r265:79063, Apr 16 2010, 13:57:41)
[GCC 4.4.3] on linux2
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface.  This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 2.6.5
>>> 4+4
8
>>> ===== RESTART =====
>>>
Hello, world!
>>>
```

When you run your code, it first prints the line `>>> ===== RESTART =====`, then runs your code underneath that line.

Exercise 1.2 – Printing

Write a program using `print` that, when run, prints out a tic-tac-toe board. Remember to save your program regularly, to keep from losing your work! The purpose of this exercise is to make sure you understand how to write programs using your computing environment; many students in introductory courses experience trouble with assignments not because they have trouble with the material, but because of some weird environment quirk.

Expected output:

```
| |  
-----  
| |  
-----  
| |
```

Exercise 1.3 – Variables

Recall that variables are containers for storing information. For example,

Program Text:

```
a = 'Hello, world!'  
print a
```

Output:

```
Hello, world!
```

The = sign is an assignment operator which tells the interpreter to assign the **value** ‘Hello, world!’ to the variable *a*.

Program Text:

```
a = 'Hello, world!'  
a = 'and goodbye...'  
print a
```

Output:

```
and goodbye...
```

Taking this second example, the value of *a* after executing the first line above is ‘Hello, world!’. But, after executing the second line, the value of *a* changes to ‘and goodbye...’. Since we ask the program to print out *a* only after the second assignment statement, that is the value that gets printed. If you wanted to save the values of both strings, you should change the second variable to another valid variable name, such as *b*.

Variables are useful because they can cut down on the amount of code you have to write. In `homework1.py`, write a program that prints out the tic-tac-toe board from exercise 1.2, but which uses variables to cut down on the amount of typing you have to do. Hint - how many different variables should you need?

Exercise 1.4 – Operators/Order of Operation

Python has the ability to be used as a cheap, 5-dollar calculator. In particular, it supports basic mathematical operators +, -, *, / as well as the power operator (**) and the modulus operator (%).

Program Text:

```
x = 5 + 7
print x
y = x + 10
print y
```

Output:

```
12
22
```

Note that we can use variables in the definition of other variables! Mathematical operators only work on numbers-*ints* or *floats*. Statements such as 'Hi' + 5 or '5' + 7 will not work.

Part I: Input the following sets of equations, and note the difference between *int* arithmetic and *float* arithmetic. You can do this just in your interpreter (you don't need to turn anything in for this part), but pay attention to the output!

1. $\frac{5}{2}$, $\frac{5}{2.0}$, and $\frac{5.0}{2}$ Note that as long as one argument is a float, all of your math will be floating point!
2. $7 * \left(\frac{1}{2}\right)$ and $7 * \left(\frac{1}{2.0}\right)$
3. $5 * *2$, $5.0 * *2$, and $5 * *2.0$
4. $\frac{1}{3.0}$ Note the final digit is rounded. Python does this for non-terminating decimal numbers, as computers cannot store infinite numbers! Take 6.004 to find out more about this...

Part II: In a file `homework1.py`, transcribe the following equations into Python, preserving order of operation with parenthesis as needed. Save each as the value of a variable, and then print the variable.

1. $\frac{3 \times 5}{2 + 3}$
2. $\sqrt{7 + 9} \times 2$
3. $(4 - 7)^3$
4. $\sqrt[4]{-19 + 100}$
5. $6 \bmod 4$ - If you aren't familiar with modular arithmetic, it is pretty straightforward- the modulus operator, in the expression $x \bmod y$, gives the remainder when x is divided by y . Try a couple modular expressions until you get the hang of it.

Part III: In `homework1.py`, use order of operation mathematics to create two equations that look the same (ie, have the same numbers) but evaluate to different values (due to parenthesization). Save each as the value of a variable, then print the variables.

Exercise 1.5 – User input

Do this exercise in `homework_1.py`. In this exercise, we will ask the user for his/her first and last name, and date of birth, and print them out formatted. Recall that you can get input from the user using the command `input('text')`, as shown in lecture.

Note: The function to get user input is `input`, and it turns whatever the user inputs into a string automatically. So, if the user inputs an int, or a float, you will need to use the relevant function to convert the input.

Here is an example of what this program should do:

Output:

```
Enter your first name: Chuck
Enter your last name: Norris
Enter your date of birth:
Month? March
Day? 10
Year? 1940
Chuck Norris was born on March 10, 1940.
```

To print a string and a number in one line, you just need to separate the arguments with a comma (this works for any two types within a print statement). The comma adds a space between the two arguments. For example, the lines:

```
mo = 'October'
day = '20'
year = '1977'
print mo, day, year
```

will have the output

```
October 20 1977
```

OPTIONAL: Now, for something completely different... a discussion on how to print strings, most prettily...

Note that none of the commas are in this output! To do that you want something like this:

```
print(f'{mo} {day} , {year}')
```

-

Exercise 1.6 – New Operators

Make sure that you understand how to use the operators `==`, `!=`, `<`, `>`, `<=`, `>=`, which are called *relation operators*. They work on all types, not just numbers, and return a *Boolean* (True/False) value. Remember, if you are using Booleans, to capitalize True and False! Here's an example shell session; try other examples you can think of.

```
>>> 5 >= 7
False
>>> 'abc' != 'def'
True
>>> x = 'abc'
>>> x == 'abc'
True
```

This next example is strange! Try to understand what's going on here, and ask if you're confused.

```
>>> a = True
>>> b = (5 < 7)
>>> a == b
True
```

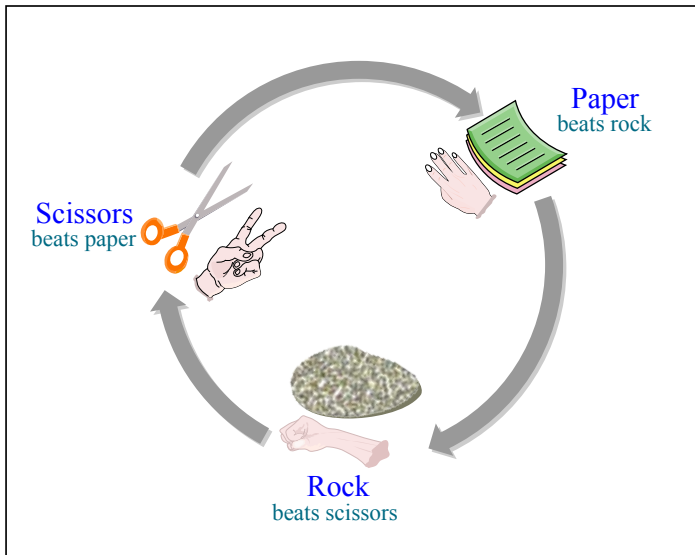
Next, the operators `+=`, `-=`, `*=`, `/=` change the value of a stored variable in a quicker way. In the following example, we add 6 to a variable in two different ways; note that we get the same result! Try using all of these operators in your interpreter window before moving on.

```
>>> x = 5
>>> x = x + 6
>>> print x
11
>>> y = 5
>>> y += 6
>>> print y
11
```

We strongly suggest you *finish all the written exercises now*, before continuing on with the next code problem.

Exercise 1.1 – Rock, Paper, Scissors

In this exercise, you are going to practice using conditionals (if, elif, else). You will write a small program that will determine the result of a rock, paper, scissors game, given Player 1 and Player 2's choices. Your program will print out the result. Here are the rules of the game:



1. First create a truth table for all the possible choices for player 1 and 2, and the outcome of the game. This will help you figure out how to code the game!

Player 1	Player 2	Result
Rock	Rock	Tie
Rock	Scissors	Player 1

2. Create a new file `rps.py` that will generate the outcome of the rock, scissors, paper game. The program should ask the user for input and display the answer as follows:

```
Player 1? rock
Player 2? scissors
Player 1 wins.
```

The only valid inputs are rock, paper, and scissors. If the user enters anything else, your program should output "This is not a valid object selection". Use the truth table you created to help with creating the conditions for your if statement(s). Read on to the next page before starting...

Note If you have a long condition in your if statement, and you want to split it into multiple lines, you can either enclose the entire expression in parenthesis, e.g.

```
if (player1 == 'rock' and
    player2 == 'scissors'):
    print 'Player 1 wins.'
```

Or, you can use the backslash symbol to indicate to Python that the next line is still part of the previous line of code, e.g.

```
if player1 == 'rock' and\
    player2 == 'scissors':
    print 'Player 1 wins.'
```

Use whichever form you feel comfortable using. When you are done coding *and testing!*, print a copy of the file and turn it in. Make sure your name and section number is in the comment section of your program.

Exercise 1.2 – For & While Loops

Create a new file called `loops.py` and use it for all parts of this exercise.

Be sure to test your code for each part before moving on to the next part.

1. Using a for loop, write a program that prints out the decimal equivalents of $1/2, 1/3, 1/4, \dots, 1/10$.
2. Write a program using a while loop that asks the user for a number, and prints a countdown from that number to zero. What should your program do if the user inputs a negative number? As a programmer, you should always consider “edge conditions” like these when you program! (Another way to put it- always assume the users of your program will be trying to find a way to break it! If you don’t include a condition that catches negative numbers, what will your program do?)
3. Write a program using a for loop that calculates exponentials. Your program should ask the user for a base `base` and an exponent `exp`, and calculate `baseexp`.
4. Write a program using a while loop that asks the user to enter a number that is divisible by 2. Give the user a witty message if they enter something that is not divisible by 2- *and make them enter a new number*. Don’t let them stop until they enter an even number! Print a congratulatory message when they **finally** get it right.

Exercise 2.0 – Print vs Return

This isn't really an exercise, just an important bit of reading. Create the file `homework2.py`. In it define these two functions:

```
def f1(x):
    print x + 1
def f2(x):
    return x + 1
```

Run this code in the shell. What happens when we call these functions?

```
>>> f1(3)
4
>>> f2(3)
4
```

It looks like they behave in exactly the same way. But they really don't. Try this:

```
>>> print f1(3)
4
None
>>> print f2(3)
4
```

In the case of `f1`, the function, when evaluated, prints 4; then it returns the value `None`, which is printed by the Python shell. In the case of `f2`, it doesn't print anything, but it returns 4, which is printed by the Python shell. Finally, we can see the difference here:

```
>>> f1(3) + 1
4
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
>>> f2(3) + 1
5
```

In the first case, the function doesn't return a value, so there's nothing to add to 1, and an error is generated. In the second case, the function returns the value 4, which is added to 1, and the result, 5, is printed by the Python read-eval-print loop.

Print is very useful for debugging. It's important to know that you can print out as many variables and strings as you want in one line, when they are separated by commas. Try this:

```
>>> x = 100
>>> print 'x:', x, 'x squared:', x*x, 'sqrt(x):', x**0.5
x: 100 x squared: 10000 sqrt(x): 10.0
```

Exercise 2.1 – Defining A Function

Recall how we define a function using `def`, and how we pass in parameters. In `homework2.py` (download this from the website if you haven't yet), transform your code from exercise 1.7 (the rock, paper, scissors game) into a function that takes parameters, *instead* of asking the user for input. Make sure to *return* your answer, rather than printing it.

For this, and all future exercises, include *at least* 3 test cases below your code.

Exercise 2.2 – Writing Simple Methods

In this problem you'll be asked to write two simple methods (method is an interchangeable term for 'function'). Be sure to *test your functions well*, including at least 3 test cases for each method. Do your work in `homework2.py`.

1. Write a method `is divisible` that takes two integers, `m` and `n`. The method returns `True` if `m` is divisible by `n`, and returns `False` otherwise. Test cases for this function are included for you; look at the conditions that they test and try to make sure your future test cases are comprehensive.
2. Imagine that Python doesn't have the `!=` operator built in. Write a method `not equal` that takes two parameters and gives the same result as the `!=` operator. Obviously, you cannot use `!=` within your function! Test if your code works by thinking of examples and making sure the output is the same for your new method as `!=` gives you.

Exercise 2.3 – Math Module

In this exercise, we will play with some of the functions provided in the `math` module. A module is a Python file with a collection of related functions. To use the module, you need to add the following line at the top of your program, right underneath the comments with your name:

```
import math
```

Example: if you want to find out what is $\sin(90^\circ)$, we first need to convert from degrees to radians and then use the `sin` function in the `math` module:

```
radians = (90.0 / 360.0) * 2 * math.pi
print math.sin(radians)
```

For mathematical functions, you can generally call `math.func`, where `func` is whatever function you want to call. For example, if you want the sine of an angle `a` (where `a` is in radians), you can call `math.sin(a)`. For logarithms, the function `math.log(n)` calculates the natural logarithm of `n`. You can calculate the log of any base `b` (as in $\log(n)$) using `math.log(n, b)`. The math module even includes constants such as e (`math.e`) and π (`math.pi`). Documentation for the math module is available at <http://docs.python.org/release/2.6.6/library/math.html>

Many computations can be expressed concisely using the “multadd” operation, which takes three operands and computes $a * b + c$. One of the purposes of this exercise is to practice pattern-matching: the ability to recognize a specific problem as an instance of a general category of problems.

In the last part, you get a chance to write a method that invokes a method you wrote. Whenever you do that, it is a good idea to test the first method carefully before you start working on the second. Otherwise, you might find yourself debugging two methods at the same time, which can be very difficult.

1. Write a function `multadd` that takes three parameters, `a`, `b` and `c`. Test your function well before moving on.
2. Underneath your function definition, compute the following values using `multadd` and print out the result:

- `angle_test` = $\sin\left(\frac{\pi}{4}\right) + \frac{\cos(\frac{\pi}{4})}{2}$
- `ceiling_test` = $\left\lceil \frac{276}{19} \right\rceil + 2 \log_7(12)$

Hint: If you are unfamiliar with the notation $\lceil \cdot \rceil$, this represents the *ceiling* of a number. The *ceiling* of some float x means that we always “round up” x . For example, $\lceil 2.1 \rceil = \lceil 2.9 \rceil = 3.0$. Look at the math module documentation for a way to do this!

If everything is working correctly, your output should look like:

```
sin(pi/4) + cos(pi/4)/2 is: _____
1.06066017178
ceiling(276/19) + 2 log_7(12) is:
17.5539788165
```

3. Write a new function called `yikes` that has one argument and uses the `multadd` function to calculate the following:

$$xe^{-x} + (1 - e^{-x})$$

There are two different ways to raise e to a power- check out the math module documentation. Be sure to return the result! Try `x=5` as a test; your answer should look like:

`yikes(5)` is 1.0303150673.

Exercise 2.4 – More Functions

Here's two more functions to try your hand at...

1. Write a method `rand divis_3` that takes no parameters, generates and prints a random number, and finally returns `True` if the randomly generated number is divisible by 3, and `False` otherwise. For this method we'll use a new module, the `random` module. At the top of your code, underneath `import math`, add the line `import random`. We'll use this module to generate a random integer using the function `randint`, which works as follows:

```
random.randint(lo, hi)
```

where `lo` and `hi` are integers that tell the code the range in which to generate a random integer (this range is inclusive). 0 to 100 is probably a decent range.

2. Write a method `roll_dice` that takes in 2 parameters - the number of sides of the die, and the number of dice to roll - and generates random roll values for each die rolled. Print out each roll and then return the string "That's all!" An example output:

```
>>> roll_dice(6, 3)
4
1
6
That's all!
```

Exercise 2.5 – Quadratic Formula

Write a function `roots` that computes the roots of a quadratic equation. Check for complex roots and print an error message saying that the roots are complex.

Hint 1: Your function should take three parameters- what are they?

Hint 2: We know the roots are complex when what condition about the discriminant is met?

Be sure to use a variety of test cases, that include complex roots, real roots, and double roots.

Exercise 2.6 – The game of Nims/Stones

In this game, two players sit in front of a pile of 100 stones. They take turns, each removing between 1 and 5 stones (assuming there are at least 5 stones left in the pile). The person who removes the last stone(s) wins.

Download `nims.py` from the website and open it up. Check out the lines of text in between the sets of `'''`, underneath the definition of `play nims`. This is called a docstring, and is handy to use to tell users of your program what parameters to pass in, and what your program does.

In this problem, you'll write a function to play this game; we've outlined it for you. It may seem tricky, so break it down into parts. Like many programs, we have to use nested loops (one loop inside another). In the outermost loop, we want to keep playing until we are out of stones. Inside that, we want to keep alternating players. You have the option of either writing two blocks of code, or keeping a variable that tracks the current player. The second way could be slightly trickier, but it's definitely do-able!

Finally, we might want to have an innermost loop that checks if the user's input is valid. Is it a number? Is it a valid number (e.g. between 1 and 5)? Are there enough stones in the pile to take off this many? If any of these answers are no, we should tell the user and re-ask them the question.

As always, feel free to ask the LAs for help on any part of this problem.

If you choose to write two blocks of code, the basic outline of the program should be something like this:

```
while [pile is not empty]:
    while [player 1's answer is not valid]:
        [ask player 1]
        [execute player 1's move]

    [same as above for player 2]
```

Be careful with the validity checks. Specifically, we want to keep asking player 1 for their choice as long as their answer is not valid, BUT we want to make sure we ask them at least ONCE. So, for example, we will want to keep a variable that tracks whether their answer is valid, and set it to False initially.

When you're finished, test each other's programs by playing them!

Exercise 2.7 – Working With Lists

Download `strings_and_lists.py` from the course website. Study the function `sum all`.

`sum all` takes a list of numbers as a parameter (note how we specify, with a comment, what the type of the parameter must be), and returns the sum of all the numbers in the list.

Now make a new function `cumulative sum` that modifies `sum all` so that instead of returning the sum of all the elements, it returns the cumulative sum; that is a new list where the i element is the sum of the first $i + 1$ elements from the original list. For example, the cumulative sum of `[4, 3, 6]` is `[4, 7, 13]`.

Exercise 2.8 – Report Card with GPA

Write a function `report_card` where the user can enter each of his grades, after which the program prints out a report card with GPA. Remember to ask the user how many classes he took (*think* - why would we need to ask this? Could we write the program a different way, which wouldn't need that info?). Example output is below.

```
>>> report_card()
How many classes did you take? 4
What was the name of this class? 18.02 -
What was your grade? 94
...
REPORT CARD:
18.02 - 94
21H.601 96
8.01 91
5.111 - 88
Overall GPA 92.25
```

Hints: You'll want to use a for loop, and you'll probably want to keep track of names and grades separately; there are a couple ways to do this. Remember, add to lists with `my_list.append(elt)`.

Exercise 2.9 – Pig Latin

Write a function `pig latin` that takes in a single word, then converts the word to Pig Latin. To review, Pig Latin takes the first letter of a word, puts it at the end, and appends “ay”. The only exception is if the first letter is a vowel, in which case we keep it as it is and append “hay” to the end.

E.g. “boot” → “ootbay”, and “image” → “imagehay”.

It will be useful to define a list at the top of your code file called `VOWELS`. This way, you can check if a letter `x` is a vowel with the expression `x in VOWELS`. Remember - to get a word except for the first letter, you can use `word[1:]`.

Be sure to look at the first optional problem for ways to improve on your Pig Latin converter.