ASSIGNMENT 2

COMP202, Fall 2020

Due: Wednesday, October 28^{th}, 11:59pm

Please read the entire PDF before starting. You must do this assignment individually.

Question 1:	15 points
Question 2:	40 points
Question 3:	45 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment, in some cases through automated tests. While these tests will never be used to determine your entire grade, they speed up the process significantly, which allows the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks. :)

Up to 30% can be removed for bad indentation of your code as well as omitting comments, or poor coding structure.

To get full marks, you must:

- Follow all directions below.
  - In particular, make sure that all file names and function names are **spelled exactly** as described in this document. Otherwise, a 50% penalty will be applied.
- Make sure that your code runs.
  - Code with errors will receive a very low mark.
- Write your name and student ID as a comment at the top of all `.py` files you hand in.
- Name your variables appropriately.
  - The purpose of each variable should be obvious from the name.
- Comment your work.
  - A comment every line is not needed, but there should be enough comments to fully understand your program.
- Avoid writing repetitive code, but rather call helper functions! You are welcome to add additional functions if you think this can increase the readability of your code.
- Lines of code should NOT require the TA to scroll horizontally to read the whole thing. Vertical spacing is also important when writing code. Separate each block of code (also within a function) with an empty line.

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions. You are responsible for knowing all of the material in these questions.*

Warm-up Question 1 (0 points)

Write a function `swap` which takes as input two int values `x` and `y`. Your function should do 3 things:

1. Print the value of `x` and `y`
2. Swap the values of the variables `x` and `y`, so that whatever was in `x` is now in `y` and whatever was in `y` is now in `x`
3. Print the value of `x` and `y` again.

For example, if your function is called as follows: `swap(3,4)` the effect of calling your method should be the following printing

```
inside swap: x is:3 y is:4
inside swap: x is:4 y is:3
```

Warm-up Question 2 (0 points)

Consider the program you have just written. Create two global integer variables in the main body of your program. Call them `x` and `y`. Assign values to them and call the swap function you wrote in the previous part using `x` and `y` as input parameters.

After calling the `swap()` function —inside the main body— print the values of `x` and `y`. Are they different than before? Why or why not?

Warm-up Question 3 (0 points)

Create a function called `counting` that takes as input a positive integer and counts up to that number. For example:

```
>>> counting(10)
Counting up to 10: 1 2 3 4 5 6 7 8 9 10
```

Warm-up Question 4 (0 points)

Modify the last function by adding an additional input that represents the step size by which the function should be counting. For example:

```
>>> counting(25, 3)
Counting up to 25 with a step size of 3: 1 4 7 10 13 16 19 22 25
```

Warm-up Question 5 (0 points)

Write a function `replace_all` which takes as input a string and two characters. If the second and third input string do not contain exactly one character the function should raise a `ValueError`. Otherwise, the function returns the string composed by the same characters of the given string where all occurrences of the first given character are replaced by the second given character. For example, `replace_all("squirrel", "r" , "s")` returns the string `"squissel"`, while `replace_all("squirrel", "t", "a")` returns the string `"squirrel"`. Do not use the method `replace` to do this.

Warm-up Question 6 (0 points)

Write a module with the following global variables:

```
lower_alpha = "abcdefghijklmnopqrstuvwxyz"
upper_alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

In this module write a function `make_lower` which takes a string as input and returns a string containing the same characters as the input string, but all in lower case. For example, `make_lower("AppLE")` returns the string `"apple"`. Do not use the method `lower` to do this. Hint: note that characters from the English alphabet appear in the same position in the two global variables.

Warm-up Question 7 (0 points)

Create a function called `generate_random_list` which takes an input an integer `n` and returns a list containing `n` random integers between 0 and 100 (both included). Use `random` to do this.

Warm-up Question 8 (0 points)

Write a function `sum_numbers` which takes as input a list of integers and returns the sum of the numbers in the list. Do not use the built-in function `sum` to do this.

Part 2

The questions in this part of the assignment will be graded.
The main learning objectives for this assignment are:

- Correctly define and use simple functions.
- Solidify your understanding of the difference between `return` and `print`.
- Generate and use random numbers inside a program.
- Understand how to test functions that contain randomness
- Correctly use loops and understand how to choose between a while and a for loop.
- Solidify your understanding of how to work with strings: how to check for membership, how to access characters, how to build a string with accumulator patterns.
- Begin to use very simple lists.
- Create a program with more than one module.
- Learn how to use functions you have created in a different module.

Note that the assignment is designed for you to be practicing what you have learned in the videos up to and including Week 7.2. For this reason, you are NOT allowed to use anything seen after Week 7.2 or not seen in class at all. You will be heavily penalized if you do so.

For full marks on the following three questions, in addition to the points listed on page 1, make sure to add the appropriate documentation string (docstring) to *all* the functions you write. The docstring must contain the following:

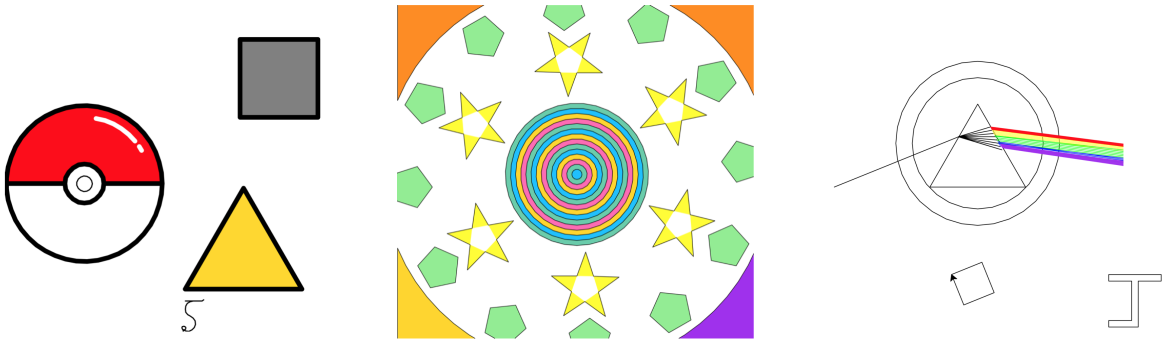
- The type contract of the function.
- A description of what the function is expected to do.
- At least 3 examples of calls to the function (except when the function has only one possible output, in which case you can provide only one example). You are allowed to use *at most* one example per function from this pdf.

Examples

For each question, we provide several **examples** of how your code should behave. All examples are given as if you were to call the functions from the shell.

When you upload your code to codePost, some of these examples will be run automatically to check that your code outputs the same as given in the example. However, **it is your responsibility to make sure your code/functions work for any inputs, not just the ones shown in the examples.** When the time comes to grade your assignment, we will run additional, private tests that may use inputs not seen in the examples.

Furthermore, please note that your code files for this question and all others **should not contain any function calls in the main body of the program** (i.e., outside of any functions). Code that does not conform in this manner will automatically fail the tests on codePost and **be heavily penalized**. It is OK to place function calls in the main body of your code for testing purposes, but if you do so, make certain that you remove them before submitting. Please review what you have learned in video 5.2 if you'd like to add code to your modules which executes only when you run your files.



Question 1: Turtle Art (15 points)

This question asks you to draw a picture by writing code with the Turtle module in the file `artwork.py`. You are free to decide the precise shape(s) and layout for your picture, but your code/drawing must satisfy at least the following requirements:

- the drawing must include at least three shapes
- at least one shape must be drawn using a for loop
- at least one shape should be drawn using a function, with the function having at least two parameters that modify the shape being drawn in some way
- the program should ask the user to enter an input value, which must then be used somehow in the drawing (e.g., the input could determine the size of the shapes, number of shapes, or the amount of space between them, amongst other things).
- the drawing must include at least two different colors
- at least one random number must be used to create the drawing in some way
- everything should fit into the standard Turtle window (without having to make the window larger)
- the first letter of your first name should appear somewhere (you must sign your artwork!)

Any submission meeting these requirements will obtain full marks, but you are encouraged to go beyond them. The most creative submissions (as judged by our TAs) will be shown in class. You are encouraged to get into the Halloween spirit and make some spooky drawings! 👻🎃

Note: Recall that you can import the `speed` function from turtle and then call `speed("fastest")` to speed up drawing, so that you don't waste time when testing your code.

Some submissions from students of previous years (with their size scaled down to fit) can be found at the top of this page.

Question 2: Algonquin Words (40 points)

Most Indigenous languages in Canada are considered endangered, and indigenous peoples are fighting for the survival of their language and culture. Indigenous languages are an integral part of Indigenous identity and Canada's cultural identity. An essential component of Indigenous decolonization and empowerment is the protection and enhancement of their heritage language. Moreover, research reveals that "children not only learn better in their own heritage language as opposed to one of the societally dominant languages, but also develop a more positive and healthier view of themselves."[1]

There are nine Indigenous languages still used in Quebec: Montagnais, Naskapi, Inuktitut, Cree, Algonquin, Atikamekw, Mohawk, Abenaki, and Mi'kmaq.[2] For this question we will write a program that allows us to retrieve the pronunciation of an Algonquin word. For the purpose of the assignment we might simplify some of the language, but you can find additional information on the Algonquin language here: `http://www.native-languages.org/algonquin.htm`

Consonants

There are 15 consonants in the language. Most of them are pronounced like the English version. There are only two exceptions we need to handle.

Character	Pronunciation
b, c, d, g, h, k, m, n, p, s, t, w, y, z	Pronounced like the English versions
j	Like the *ge* sound at the end of the English word *mirage*
dj	Like *j* in *jar*.

Vowels

There are 4 vowels in the language all of which could appear with a grave accent.

Character	Pronunciation
a, à	Pronounced like *a*
e, è	Pronounced like *e*
i	Pronounced like *i*
ì	Pronounced like *ee*
o	Pronounced like *u*
ò	Pronounced like *o*

Diphthongs

There are 6 diphthongs in the language. These are special pairs of characters which produce a distinctive sound.

Character	Pronunciation
aw	Pronounced like *ow* in *cow*
ay	Pronounced like *eye*
ew	Pronounced like *ao*
ey	Pronounced like *ay* in *hay*
iw	Pronounced like *ew*
ow	Pronounced like *ow* in *show*

Algonquin Utility Functions

Let's start by creating a module named `algonquin_utils` which contains several helper functions needed to implements the full program. Inside this module add the following global variables:

[1] https://www.erudit.org/fr/revues/du/2008-du2547/019562ar.pdf

[2] Dorais, L. J. (1996). The Aboriginal languages of Quebec, past and present. In J. Maurais (Ed.), Quebec's Aboriginal Languages: History, Planning, Development (pp. 43-100). Clevedon, England: Multilingual Matters Ltd.

```
CONSONANTS = "bcdghkmnpstwyzj"
VOWELS = "aeio"
VOWELS_WITH_ACCENT = "àèìò"
PUNCTUATION = ',;:.?!-'
DIPHTHONGS = ['aw', 'ay', 'ew', 'ey', 'iw', 'ow']
```

For full marks, all the following functions must be part of this module:

- `is_valid_consonant`: given a string, determines if it's a single character representing a valid consonant in Algonquin. This function should not be case sensitive. Please note that `"dj"` is not a single character.

  For example:

  ```
  >>> is_valid_consonant('j')
  True
  >>> is_valid_consonant('l')
  False
  >>> is_valid_consonant('G')
  True
  """
  ```

- `is_valid_vowel`: given a string, determines if it's a single character representing a valid vowel in Algonquin. This function should not be case sensitive.

  For example:

  ```
  >>> is_valid_vowel('a')
  True
  >>> is_valid_vowel('ai')
  False
  >>> is_valid_vowel('h')
  False
  ```

- `is_valid_single_word`: given a string, determines if it contains a single word made up by valid letters in Algonquin. This function should not be case sensitive.

  For example:

  ```
  >>> is_valid_single_word('Kwey')
  True
  >>> is_valid_single_word('rats')
  False
  >>> is_valid_single_word('ay,dj')
  False
  ```

- `is_valid_phrase`: given a string, determines if it contains only valid letters in Algonquin, accepted punctuation marks, or space characters. This function should not be case sensitive. For example:

  ```
  >>> is_valid_phrase('Kwey') # Hello
  True
  >>> is_valid_phrase('Andi ejayan?') # Where are you going?
  True
  >>> is_valid_phrase('I scream, you scream, we all scream for ice cream')
  False
  ```

Algonquin Pronunciation

Let's now create a module named `algonquin_pronunciation`. This module will contain the functions that determine the pronunciation of an Algonquin word. **Note that for this question you are not allowed to use the replace method.** As always you can add helper functions if you want to. Please make sure to reduce code repetition as much as possible.

Note that all of the functions in this module that take a string as input and return its pronunciation should not be case sensitive. Also, the returned pronunciation string should contain only upper case letters.

For full marks, all the following functions must be part of this module:

- `get_consonant_pronunciation`: given a string, if it is a valid consonant return its pronunciation (see tables above for details on the pronunciation), otherwise return an empty string.

  For example:

  ```
  >>> get_consonant_pronunciation('d')
  'D'
  >>> get_consonant_pronunciation('j')
  'GE'
  >>> get_consonant_pronunciation('r')
  ''
  ```

- `get_vowel_pronunciation`: given a string, if it is a valid vowel return its pronunciation (see tables above for details on the pronunciation), otherwise return an empty string.

  For example:

  ```
  >>> get_vowel_pronunciation("a")
  'A'
  >>> get_vowel_pronunciation("è")
  'E'
  >>> get_vowel_pronunciation("o")
  'U'
  ```

- `get_diphthong_pronunciation`: given a string, if it is a valid diphthon return its pronunciation (see tables above for details on the pronunciation), otherwise return an empty string.

  For example:

  ```
  >>> get_diphthong_pronunciation("ay")
  'EYE'
  >>> get_diphthong_pronunciation("ow")
  'OW'
  >>> get_diphthong_pronunciation("oy")
  ''
  ```

- `get_word_pronunciation`: given a string, if it is a valid word return its pronunciation, otherwise return an empty string.

  For example:

  ```
  >>> get_word_pronunciation('kwey') # hello
  'KWAY'
  >>> get_word_pronunciation('madjashin') # see you later
  'MAJASHIN'
  ```

```
>>> get_word_pronunciation('kasagiyan') # i love you
'KASAGIYAN'
```

- `tokenize_sentence`: given a string, if it is a valid phrase break it down into strings representing either single words or a sequence of punctuation marks and space characters. The function returns a list containing all these strings. If the input string is not a valid phrase, then return an empty list.

  For example:

```
>>> tokenize_sentence("a test")
['a', ' ', 'test']
>>> tokenize_sentence("just__@a# test!")
[]
>>> tokenize_sentence('Kwey') # Hello
['Kwey']
>>> tokenize_sentence('Kwey, anin eji-pimadizin?') # Hello, how are you?
['Kwey', ', ', 'anin', ' ', 'eji', '-', 'pimadizin', '?']
```

- `get_sentence_pronunciation`: given a string, if it is a valid string return its pronunciation, otherwise return an empty string.

  For example:

```
>>> get_sentence_pronunciation('Kwey') # Hello
'KWAY'
>>> get_sentence_pronunciation('Andi ejayan?') # Where are you going?
'ANDI EGEEYEAN?'
>>> get_sentence_pronunciation('Mino ishkwa nawakwe') # Good afternoon
'MINU ISHKWA NOWAKWE'
>>> get_sentence_pronunciation("I scream, you scream, we all scream for ice cream")
''
```

Question 3: Cards (45 points)

In this question we will follow up on our card functions from Assignment 1. Make sure that you have downloaded the `card1.py` file provided to you in this assignment. The file contains the solution to the Assignment 1 Card question. You will need to import the global variables and functions from that module to help you to write the following functions. **Do not modify the card1.py file.** You will not be able to upload it to codePost.

For the rest of this question, please note that unless otherwise specified you can assume that the inputs to the functions are valid (both the type and the format will match what the function is expecting).

Write your code for the following in a file called `card2.py`.

First, we will define four global variables:

- `SUITS`: a list of the four suit integers, in the order of hearts, diamonds, clubs, spades (use the global variables already made for the suits from card1.py).
- `RANKS`: a list of the thirteen rank integers (starting at Two and ending at Ace) (use the global variables already made for the ranks).
- `SUITS_STR`: a list of strings, one for each suit name (in the order Hearts, Diamonds, Clubs, Spades). The strings should be all uppercase.
- `RANKS_STR`: a list of strings, one for each rank name (starting at Two and ending at Ace). The strings should be all uppercase.

Then, define the following functions. Remember to use the above global variables whenever possible.

- `get_card(suit, rank)`: takes two integers as arguments (one for a suit between 0 and 3, and the other for a rank between 0 and 12), and returns the integer representation of the card with that suit and rank (i.e., a number between 1 and 52).
  EXAMPLE:

```
>>> get_card(HEARTS, TWO)
1
>>> get_card(HEARTS, THREE)
5
>>> get_card(3, 12) # Ace of Spades
52
```

- `card_to_string(card)`: takes an integer as argument for a card between 1 and 52, and returns a string for that card's name in the form `RANK of SUIT`. Note that the rank and suit are capitalized, and 'of' is not capitalized.
  EXAMPLE:

```
>>> card_to_string(1)
'TWO of HEARTS'
>>> card_to_string(5)
'THREE of HEARTS'
>>> card_to_string(52)
'ACE of SPADES'
```

- `hand_to_string(hand)`: takes a list of cards between 1 and 52 as argument, and returns a string containing the names of all the cards, each card name being separated by a comma and a space.
  EXAMPLE:

```
>>> hand_to_string([1, 2, 3, 4])
'TWO of HEARTS, TWO of DIAMONDS, TWO of CLUBS, TWO of SPADES'
```

```
>>> hand_to_string([52])
'ACE of SPADES'
```

- `get_deck()`: returns a list of integers containing the 52 cards in a deck of playing cards (one card of each suit and rank). The list does not have to have a specific ordering of cards, as long as all cards are present.
  EXAMPLE:

```
>>> deck = get_deck()
>>> len(deck)
52
```

- `all_same_suit(cards)`: takes a list of cards between 1 and 52 as argument, and returns `True` if all cards in the list are of the same suit, and `False` otherwise.
  EXAMPLE:

```
>>> all_same_suit([4, 52])
True
>>> all_same_suit([1, 2, 3, 4])
False
```

- `all_same_rank(cards)`: takes a list of cards between 1 and 52 as argument, and returns `True` if all cards in the list are of the same rank, and `False` otherwise.
  EXAMPLE:

```
>>> all_same_rank([4, 52])
False
>>> all_same_rank([1, 2, 3, 4])
True
```

We will now start a new file called `game.py` for the following functions. These functions will allow us to play a simple card game in Python.

- `calculate_winner(points)`: takes a list of scores (integers) as argument, and returns a list containing the indices of the list corresponding to the lowest points in the `points` list. (Any ordering of the indices is fine.) E.g., if the `points` list is `[100, 5, 20, 42]`, then the function should return `[1]`, as the lowest number in the list (5) was at index 1.
  EXAMPLE:

```
>>> calculate_winner([100, 5, 20, 42])
[1]
>>> calculate_winner([100, 5, 20, 5])
[1, 3]
```

- `calculate_round_points(hand)`: takes a player's hand (list of cards, i.e. a list of integers between 1 and 52) as argument, and returns the point value of that hand. Points for a hand are calculated depending on the ranks of the cards in the hand. An Ace is 1 point; Twos through Tens are their numeric value, and Jack/Queen/King are all 10 points.
  EXAMPLE:

```
>>> calculate_round_points([1, 2, 3, 4])
8
>>> calculate_round_points([49, 50, 51, 52])
4
```

- `is_valid_group(cards)`: takes a list of cards (i.e. a list of integers between 1 and 52) as argument, and returns `True` if the cards form a valid group, and `False` otherwise. A group is a set of three or more cards of the same rank (e.g., two of hearts, two of diamonds, two of clubs).
  EXAMPLE:

```
>>> is_valid_group([1, 2, 3])
True
>>> is_valid_group([1, 2, 3, 52])
False
```

- `is_valid_sequence(cards)`: takes a list of cards (i.e. a list of integers between 1 and 52) as argument, and returns `True` if the cards form a valid sequence, and `False` otherwise. A sequence is a set of three or more cards of the same suit with consecutive rank (e.g., two of hearts, three of hearts, four of hearts). EXAMPLE:

```
>>> is_valid_sequence([1, 5, 9])
True
>>> is_valid_sequence([1, 5, 10])
False
>>> is_valid_sequence([30, 34])
False
>>> is_valid_sequence([34, 38, 30])
True
```

In this card game there can be many players in a game. A round consists of each player taking a turn in order until one player declares they have won. During a player's turn, they draw a card (from the stock or discard pile), and then discard a card from their hand. A player can declare they have won the game when, after discarding a card, they can arrange their hand into a combination of groups and sequences (as defined above).

To represent a player, we will create a file with two functions: one to choose where to draw from, and one to choose which card to discard from the hand.

We will first write a player who plays completely randomly. In a file `random_player.py`, write the following functions:

- `draw(hand, top_discard_card)`: Takes a list of the cards in the player's hand as argument, as well as the card currently at the top of the discard pile (if there is no card in the discard pile, `None` is given instead). Returns either `'stock'` or `'discard'` at random (unless there is no top card in the discard pile, in which case only `'stock'` should be returned.
  EXAMPLE:

```
>>> import random_player
>>> random_player.draw([4, 50, 15, 21], 5)
'stock' # or 'discard', randomly
>>> random_player.draw([4, 50, 15, 21], None)
'stock'
```

- `discard(hand)`: Takes a list of the cards in the player's hand as argument, and returns a random card in the hand. Make sure that your function does not modify the input list.
  EXAMPLE:

```
>>> import random_player
>>> random_player.discard([4, 50, 15, 21])
50 # or 4, 15, or 21
```

Next we will write in a file `human_player.py` the same functions, but this time asking the user for input instead of randomly deciding:

- `draw(hand, top_discard_card)`: Takes a list of the cards in the player's hand as argument, as well as the card currently at the top of the discard pile (if there is no card in the discard pile, `None` is given instead). Asks the user to enter either `'stock'` or `'discard'` and returns that string.
  EXAMPLE:

```
>>> import human_player
>>> location = human_player.draw([4, 50, 15, 21], 5)
Draw location: stock
>>> print(location)
stock
```

- `discard(hand)`: Takes a list of the cards in the player's hand as argument. Prints out each card and its index in the list. Asks the user to enter one of the indices, and returns the card at that index.
  EXAMPLE:

```
>>> import human_player
>>> card_to_discard = human_player.discard([4, 50, 15, 21])
1          TWO of SPADES
2          ACE of DIAMONDS
3          FIVE of CLUBS
4          SEVEN of HEARTS
Choice: 3
>>> print(card_to_discard)
15
```

When we grade this question we will test each function individually to make sure it operates as expected. But these functions will not let you play the card game by themselves. There would need to be extra code added that calls all these functions as appropriate. We may return to this idea in a later assignment to finish up our card game.

What To Submit

You must submit all your files on codePost (https://codepost.io/). The file you should submit are listed below. Any deviation from these requirements may lead to lost marks.

`artwork.py`
`algonquin_utils.py`
`algonquin_pronunciation.py`
`card2.py`
`game.py`
`random_player.py`
`human_player.py`
`README.txt` In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your program, it may lead the TA to give you more partial credit.
Remember that this assignment like all others is an `individual` assignment and must represent the entirety of your own work. You are permitted to verbally discuss it with your peers, as long as no written notes are taken. If you do discuss it with anyone, please make note of those people in this `README.txt` file. If you didn't talk to anybody nor have anything you want to tell the TA, just say "nothing to report" in the file.