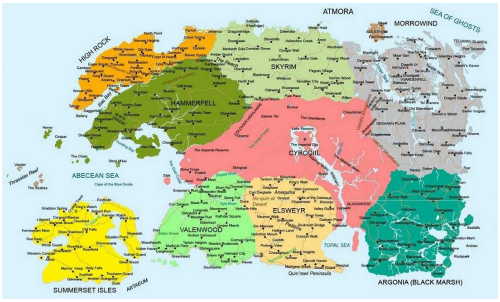
Computer Science I
MeteoTurtle

CSCI-141
Lab 5

09/15/2022



1 Problem

News programs in Simland use the *MeteoTurtle* program to present weather. MeteoTurtle is an interpreter for a 'shorthand language' that expresses turtle programs to represent a MeteoTurtle weather description compactly as a *string*. **This interpreter is defined by a function that takes a string argument.**

The content of the string is a *sequence of commands* shown below. The MeteoTurtle commands cause the turtle to draw or change its state.

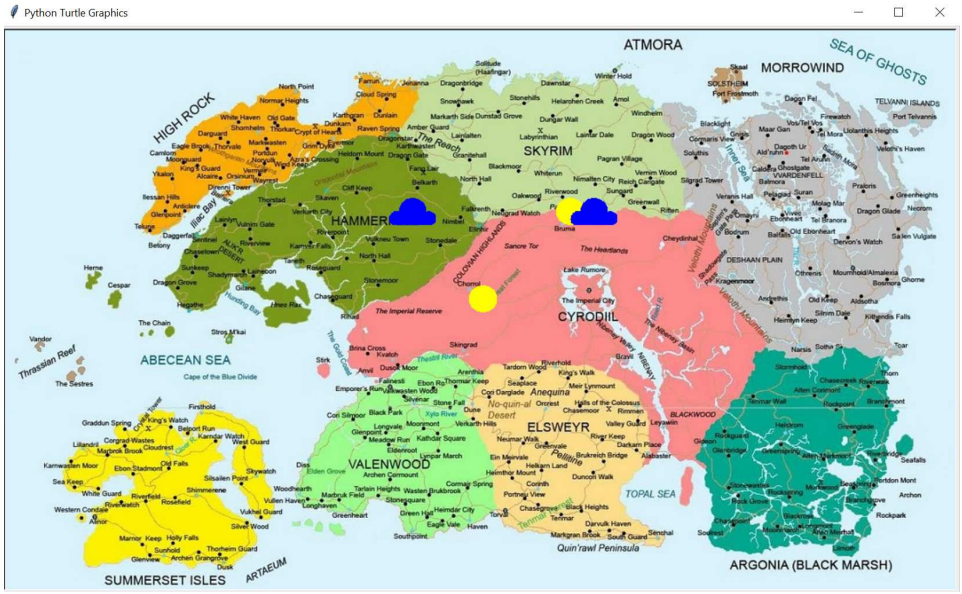
Table 1: MeteoTurtle Commands

Command	Effect
`S`	Draw a sun.
`P`	Draw a sun partially covered by a cloud.
`C`	Draw a cloud.
`R`	Draw a cloud and rain.
`W`	Draw a cloud and 3 snowflakes.
`T#`	Write temperature value in Fahrenheit on a white background. Examples: `T-10`, `T55`.
`A#`	Make a red circle with radius # to represent areas with weather alert. Examples: `A15`, `A5`.
`G#x,#y`	GoTo location #x, #y, #x and #y are integers; Examples: `G28,-43`, `G10,34`.

Pre and post-conditions for all weather icons and commands are that the turtle is facing East and has the pen up. All MeteoTurtle instructions begin at the current turtle location established by the initial default or by execution of the previous command. All numerical arguments (#, #x, #y) can be one or more numeric digits.

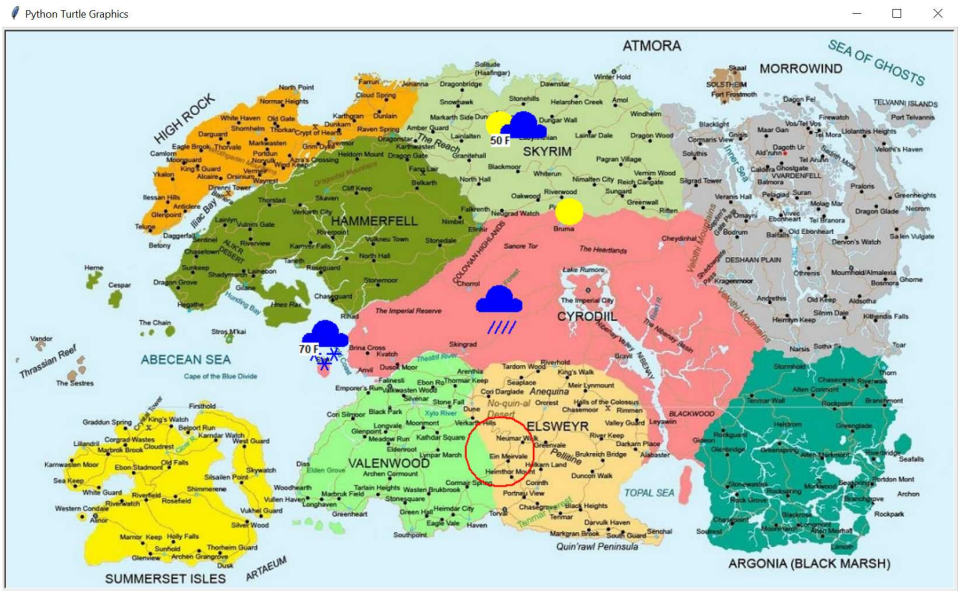
When the MeteoTurtle program starts, it loads the map image and places the turtle at the origin, facing East and with the turtle pen up. Then the program prompts to obtain the command string. Finally it interprets the string to move and draw the weather symbols.

Simland's world limit coordinates are -550 W, 550E, 325N, -325 S. Below are some sample MeteoTurtle programs and the command strings that produced the graphics.



Command string entered: `SG100,100PG-100,100C`

Command string entered: `WG-500,0WG500,0S`



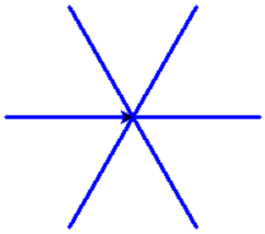
Command string entered:

`RG100,100SG20,200PG10,190T50G-200,-40WG-210,-50T70G20,-200A40`

2 Problem-solving Session (15%)

Work in teams as determined by the instructor during problem-solving. Number your answers to each problem-solving question.

1. Use iteration to write a Python function that draws a snowflake with "arms" of length `size` from its center:



2. Write a Python function `get_number` that has a string parameter type and extracts a number from the digits at the front of a string. The string argument may be the empty string. The return value must be a string and *not* a number.
   Examples: "123P456" would return "123". "-200,30P" would return "-200".

3. Explain in words, not code, an algorithm or process that would interpret a MeteoTurtle command string. A high level explanation is all that is required.

4. Write a Python function to interpret and process a string containing any combination of **the first three commands** in Table 1.

   Assume the input is a string containing one or more instances of the first three MeteoTurtle commands. Also assume that a Python function exists for each of the commands. For example, assume there is a `process_P(...)` function to interpret the `P` command. Do not write code for all the `process_S(...)`, and other functions; just *stub the ones you need* and use the stubs.
   Do not implement all of the command functions during problem-solving, and do not worry about error checking at this time.

5. Write a Python function `process_G` to properly interpret and process the GoTo command. How is GoTo different from other commands considered in problem-solving? What special, extra work is needed?
   Example: G100,50 eventually would issue `turtle.goto(100,50)`.

Notify the instructor or the SLIs when you finish.

3 In-lab Session (10%)

After the end of the problem-solving session, begin work independently to complete the following tasks for upload to the In-lab MyCourses dropbox:

1. Download:

   `https://www.cs.rit.edu/~csci141/Labs/05/files.zip`

2. Create a new project and folder (e.g. **lab05**), and unpack **simland.png** and **meteo.py** from the zip file into the project. You will complete function stubs in `meteo.py` and use the graphic as the canvas background. See the downloaded files.

3. Create a new file, named `meteo_turtle.py`, and write the Python function to interpret and process a string containing a sequence of MeteoTurtle commands. Start by processing only the first three commands for the in-lab; later you will implement the rest for the full solution on your own.

4. Write a `main` function that prompts for a string containing commands and calls the function that interprets and processes the command string.

5. Write a Python function `get_number` that takes a string and returns a string of the digits at the front of the string. The function returns `None` if the string does not start with a number. The string argument may be the empty string.

For the In-lab, zip your `meteo_turtle.py` work-in-progress to a file named **lab05.zip** and upload it to mycourses. Do this at any time before the In-lab deadline.

4 Implementation (75%)

Write the remaining functions in the provided file `meteo.py`.

Import the `meteo.py` module into your `meteo_turtle.py` file, to initialize the background and draw weather icons. **Do not copy those functions into `meteo_turtle.py`.** Implement all MeteoTurtle commands in Table 1 for the complete program.

For a nice look use:

- `r=16` for the radius of the sun;
- `size=8` for the arm of a snowflake;
- font "Arial", size 9, "bold" for the temperature; and
- a white rectangle of sides 36×16 for the temperature's background.

The rain drawing function that uses `draw_cloud` does not have to produce exactly the same weather icon, but the icon should be similar. You need a function to draw a white rectangle of sides 36×16 for the temperature's background.

4.1 Details of Operation

The `main` function prompts the user to enter a string containing MeteoTurtle commands. It then calls the MeteoTurtle interpret function passing that string in for processing. Assume that the user enters exactly one string containing a sequence of MeteoTurtle commands. If there are no errors, the `main` function waits for the user to close the drawing window using `turtle.done()`.

MeteoTurtle command shall return `None` to represent an error. The program prints the text it tried to interpret, terminate and does not process the string any further. For example: `"There was an error with T#"`.

The MeteoTurtle interpret function only needs to check the first character of the string at each iteration. It does not need to change the string, but it will need to move forward somehow to process the next command.

When it recognized a command string, it sends the substring to the proper MeteoTurtle command function. If the character is not a valid MeteoTurtle command, print an error message, and stop interpreting.

Create working functions for processing each of the MeteoTurtle commands (for example `process_P(...)` function to interpret the `P` command). When the program identifies a MeteoTurtle command, it should call the appropriate function to interpret that command substring. If it processes that command without error, then the program should advance to the next substring to interpret the next command.

It is acceptable to use iteration, slicing, and indexing of strings.

4.2 Error Handling

Each MeteoTurtle command function should handle errors. If an error occurs, your function should return `None`. For example, `process_A( "AA100" )` should return `None` because the first *A* in the string is missing its radius value.

The numbers accepted for a MeteoTurtle program are all integers. Input of a floating point number is an error.

4.3 Constraints and Restrictions

The use of the Python `re` module, or any similar library, is prohibited.

4.4 Tools and Tips

This section outlines some tools and tips for this assignment.

1. This program must check whether a character input is a numeric digit. Python has a function that checks if a string contains numeric digits. If `st` is a reference to a

string, the call `st.isdigit()` returns `True` if `st` contains only numeric digits, and False otherwise.

2. `turtle.write` writes a text message on the canvas.

5 Grading

One long function that does everything will result in a deduction of 50% of the implementation points.

The implementation portion of the assignment will be graded as follows:

- 10%: The main function displays the map, prompts the user for a string and calls an interpret function to process it as a string of MeteoTurtle commands.
- 25%: An interpreter function interprets the string argument, and calls the proper MeteoTurtle command function. Errors are handled properly, such as invalid command characters.
- 5%: The project structure has functions in two Python files, and one module correctly imports the other.
- 35%: A function exists to process each of the MeteoTurtle commands, and each function handles possible errors properly.
- **-10%** Style and Documentation:
  Points for style and documentation are now **minus points**, which means they are *taken off the top of the earned grade.*

  Docstrings should be triple-quote form (`"""`) and have a purpose statement, parameter descriptions, return type descriptions, plus pre/post-conditions where appropriate.

  Even a perfectly working implementation will be subject to grade deductions of up to 10% if that program lacks proper style, layout and documentation or if the submission did not follow submission instructions.

  The deductions below are limited to the maximum of 10%:
  - 5 points for incomplete or missing file docstring and content.
  - 5 points for incomplete or missing function docstrings.
  - 3 points for incorrect python file names.
  - 4 points for layout or style issues (e.g. function naming).
  - 10 points for zipping an entire directory instead of only the required files.

6 Submission

For submission zip `meteo_turtle.py`, `meteo.py` and `simland.png` into a file named `lab05.zip` and submit the zip to the *MyCourses assignments dropbox* by the due date.

Do not zip any other files!