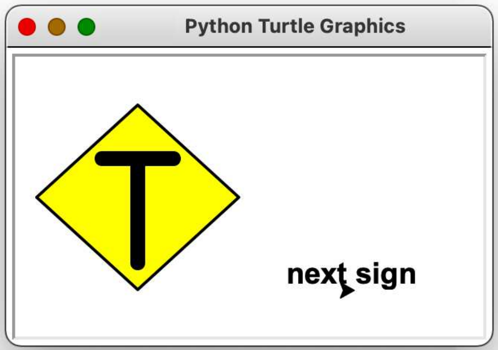
Computer Science I — CSCI-141
Road Signs — Lab 1

01/18/2023

1 Problem

Using Python's turtle graphics module, design a program that draws two different road signs using re-usable functions. The goal of this lab is to become familiar with various Python Turtle commands and use them to create a module (file) with basic Python program structure.



2 PSS: Problem-solving Session (15%)

Do not use a computer for the PSS; use the whiteboard or paper. Work in a team of students as determined by the instructor.

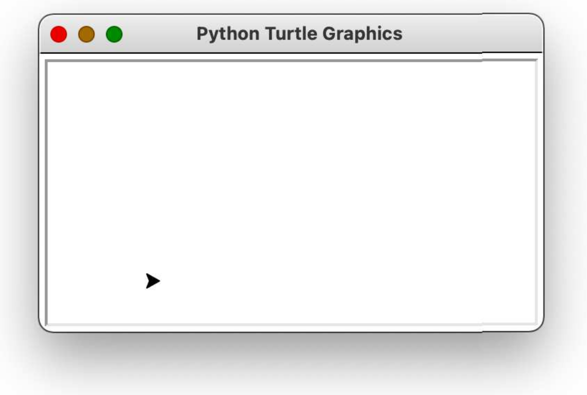
Each team has 55 minutes to complete the following activities.

1. Design how you will layout 2 signs in a row of the same size. This is best done with diagrams. Define the dimensions of a single sign and the order in which to draw the sign components. From the single sign design, determine the canvas size.
   Note: you will have to draw using relative positioning, and that means you cannot use absolute coordinate locations.
2. Write Python function(s) to draw the sign background. The turtle must begin and end in the same location and with the same orientation. (Documentation is not needed for the PSS.)
3. Write Python function(s) to draw the first complete sign. Determine how to *reuse* the function(s) defined earlier.
4. Design your own road sign. You have freedom to make this sign as simple, complex or crazy as desired. (Please keep it clean and tasteful!) Explain how you would define the drawing function using Turtle. What shapes would you use? Where does the turtle start and in what state does the turtle finish?

Each team shall number the items and deliver results at the end of problem-solving.

If your team is done early, work on designing your canvas layout.

You should set up the canvas so that it will fit the drawn figures. Below is a snapshot that shows the initialized canvas immediately before beginning to draw the first sign.



2.1 Canvas Size and Coordinate System

You should use `turtle.setup()` to set the canvas size to a width and height that fits the 2 signs. Each sign is the same size, and the width of the canvas needs to be wide enough to display 2 signs in a row with an additional margin around the top, bottom and sides.

The `turtle.setworldcoordinates()` function sets the coordinate system within the canvas window dimensions. Research the parameters in the Python turtle documentation to learn about the lower left and upper right specifications that will place the origin (0,0) in a desirable location for this assignment.

`https://docs.python.org/3.3/library/turtle.html`

Your instructor and SLI will tell you how to access the In-lab and Full lab at the end of the problem-solving session.

3 In-lab Session (10%)

The goals of In-lab are to learn the CS systems as a backup in case your system fails and to start your implementation of the lab.

The Instructor and SLI will help you learn to use the CS department systems and learn these things:

1. How to log in to CS lab systems;
2. How to start a browser and look up Python 3 documentation;
3. How to change your password with 'knockknock.cs.rit.edu';
4. How to open a terminal window for starting PyCharm, Idle or a terminal window for Python coding;
5. Where and how to save files;
6. How to contact the SLI to ask questions via email; and
7. How to get help from system administrators' via GCCIS-IT's email: gccisit@rit.edu.
8. Create your `signs.py` program module, import `turtle`, write a Python `main` function and your function to draw the sign background. Make the main function call your background function and then call `turtle.done()` to wait for the user to close the canvas window. By end of the In-lab, your program should draw something.

For the In-lab, zip your `signs.py` work in progress to a file named **lab01.zip** and upload it to mycourses. You must submit your In-lab submission before you leave the lab. Mycourse's time of submission will be considered.

4 Assignment Details

4.1 Implementation (55%)

Each student must individually implement and submit their own solution to the problem as a Python program named `signs.py`.

When run, the program must pause after drawing and wait for the user to close the turtle window. Wait for this acknowledgement by inserting a `turtle.done()` statement.

You should research how to use various turtle commands so that you can make a Python program that draws things.

The drawing must include the following items:

- A road sign indicating 'T' intersection; and
- One other road sign of your choosing.

You must fill the shapes with colors. Research turtle color fill to assist with this portion of the task.

The two signs drawn must appear in a row on the drawing canvas as shown in the earlier picture.

You must make reusable functions to draw each of the sign elements. It is a poor practice to draw everything in one long function.

Note: *You must retest code when making even the smallest of changes.*

Hints and Tips

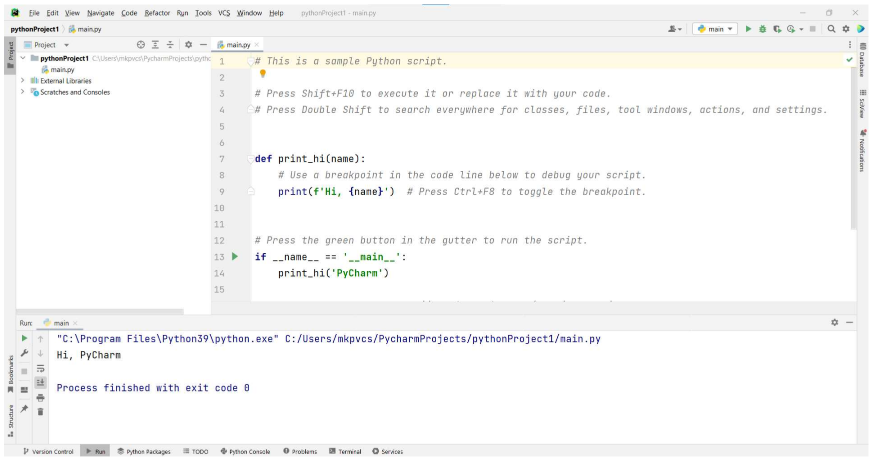
- Turtle Doc Website: `https://docs.python.org/3.3/library/turtle.html`
- `setup` and `setworldcoordinates` help with setting up the canvas.
- Turtle can use numbers or words for choosing colors.
- Design functions for each element and test them first. Then draw the larger picture using these functions.
- `https://www.cs.rit.edu/~csci141/resources.html` is the location of resource page for the course. There is also a link found near the top of the main course page.

4.2 PyCharm setup (20%)

Follow the instructions at `https://www.cs.rit.edu/~csci141/Docs/python-setup.html` to configure PyCharm to use the Python System Interpreter as the default interpreter.

Run the default Python program in PyCharm and make a screenshot of your screen when you complete running the program (one screenshot is sufficient). You will include this with your submission.

The screenshot shall look similar to the following but you have to put your name on top of the file:



```python
# This is a sample Python script.

# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool windows, actions, and settings.


def print_hi(name):
    # Use a breakpoint in the code line below to debug your script.
    print(f'Hi, {name}')  # Press Ctrl+F8 to toggle the breakpoint.


# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    print_hi('PyCharm')

```

```
"C:\Program Files\Python39\python.exe" C:/Users/mkpvcs/PycharmProjects/pythonProject1/main.py
Hi, PyCharm

Process finished with exit code 0
```

How to take a screenshot

The task of taking a screenshot is different in the various operating systems. Below is a list of the operating systems and how to take a screenshot in each.

1. Windows

   In Windows there are two ways to take a screenshot.

(a) Using the print screen button. When you are ready to take a screenshot, press the “Print Screen” (PrtSc) button on your keyboard. This will capture an image of your entire desktop. You must then open an image editing application, such as Paint, and paste the image into the canvas. Remember this is a screenshot of your entire desktop and may contain information you do not want to share. You can crop the image to show only the parts you want to share. Save the image, and you have your screenshot.

(b) Using the Snipping Tool. This is easier than using the print screen button. Start the snipping tool application and select New. This will gray out your screen and give you a crosshair cursor. Click and hold the mouse button at the corner of the area you wish to capture, then drag to the opposite corner and release. This will capture the image in the Snipping Tool. Then save the image and you have your screenshot.

2. Linux

   There are several ways to take a screenshot in Linux. Let’s use Ubuntu as an example, but most of these will work on any Linux distro. If you aren’t using GNOME, then GNOME-specific items won’t work.

   (a) Using the print screen button. You can also take a screenshot of the entire screen by pushing the “Print Screen” (PrtSc) button on your keyboard. To get a screenshot of only the active window, use Alt-PrtSc. A dialog will appear asking you to save the image. Save the image and you have your screenshot. Be aware that the image may need some editing if you wish to not show everything contained in it.

   (b) Using gnome-screenshot. This can be done on the command line. Open a terminal and enter “gnome-screenshot”. This will take a screenshot and open a dialog to save it. Save it and you have your screenshot. Be aware that the image may need some editing if you wish to not show everything contained in it.

3. macOS (Macintosh)

   There are several ways in Macos to take a screenshot. A couple are outlined below.

   (a) Entire Screen. To take a screenshot of the entire screen press “Command + Shift + 3”. This will place the screenshot image on your desktop in a .png format. Be aware that the image may need some editing if you wish to not show everything contained in it.

   (b) Part of the screen.

      i. Press Command+Shift+4. You’ll see that your cursor changes to a crosshair pointer.

      ii. Move the crosshair pointer to where you want to start the screenshot.

      iii. Drag to select an area. To adjust the area, hold Shift, Option, or the Space bar while you drag.

      iv. When you’ve selected the area you want, release your mouse or trackpad button. Or to cancel, press the Escape (esc) key.

      v. Find the screenshot as a .png file on your desktop.

5 Grading

- 15%: PSS
- 10%: In-lab session and submission.
- 20%: Sign-drawing Behavior
  - 10%: Draws two distinct signs filled with the appropriate color.
  - 5%: Drawings fit within the canvas window.
  - 5%: Displays the drawing and waits for the user to close the canvas and cause the program to exit.
- 10%: Function Design and implementation. Drawing the entire sign in one long function or with no helper functions is bad design and subject to a 50% deduction.
  - 5% A function that draws the sign background for each sign
  - 5% Functions that draw the complete signs.
- 10%: Style and Documentation
  - 5%: Follows the code style guidelines on the course web site. The code should be commented well enough so that it would be easy for someone to reuse functions to design another sign.
  - 5%: A *docstring* at the top of the source file module includes at least the author's full name. This is a *file documentation block* that should use *triple-quote* syntax as shown in the course examples.
- 20%: Screenshot showing the correct configuration of your PyCharm.
- 15%: Submission used zip properly and uploaded correctly. (See below.)

6 Submission

Failure to follow these instructions will result in a 10% deduction.

- Make sure your name is included in all submitted files.
- Create a document that will include your name and the screenshot.
- Convert the document to **PDF** format for submission.
  Name this document `configuration.pdf`.
- Create a zip file named `lab01.zip` containing the `signs.py` file and `configuaration.pdf` file.
- Use *only the zip program*. Do not use rar, 7zip or other compression tools.
- Submit your zip file to the appropriate myCourses assignment dropbox.

Check to make sure that the zip file contains ONLY `signs.py` and `configuaration.pdf` files