Assignment 4 — V8

Due date: Friday, April 8, 11:59 pm
Submission via Git only

Programming environment

For this assignment you must ensure your code executes correctly on the *reference platform* (i.e., computers on ELW B238, which can be accessed using ssh) you configured as part of Lab 01. This same environment will also be used by the teaching team when evaluating your submitted work. You will have to make sure that your program executes perfectly on the reference platform. If your programs do not run on reference platform, your submission will receive 0 marks.

All sample code for this assignment are available in the 'a4' folder of your git repository and you must use the git pull command to download a copy of the files: **git pull**

git pull, if you already cloned the 'assignments' branch as indicated in the Step 3 of the step-by-step guide for submissions. Of course, you'll need to be inside the repository folder to execute this command.

Individual work

This assignment is to be completed by each individual student (i.e., no group work). You are encouraged to discuss aspects of the problem with your fellow students. However, sharing of code fragments is strictly forbidden. Note SENG 265 uses highly effective plagiarism detection tools to discover copied code in your submitted work. Both, using code from others and providing code to others, are considered cheating. If cheating is detected, we'll abide by the strict UVic policies on academic integrity: https://www.uvic.ca/library/help/citation/plagiarism/

Learning objectives

I. Learn how to generate web pages or how to generate programs using Python — generate HTML and SVG code programmatically.
II. Learn object-oriented design with PART-OF or aggregation hierarchies — using Python classes, objects, and attributes (methods and data) to represent hierarchical HTML and SVG code.
III. Appreciate the software engineering concepts of abstraction, encapsulation, and separation of concerns.
IV. Learn and appreciate Python type hinting.
V. Learn how to understand application programmer interfaces (APIs).
VI. Learn how to generate random numbers in Python — random numbers are frequently employed in algorithms, simulations, and quantum computing.
VII. Learn how to manage sequential text files using the Python file I/O API.
VIII. Learn to write Pythonic code using *namedtuple*.
IX. Practice incremental software development.
X. Use Git to manage changes in your source code and annotate the evolution of your solution with messages provided during commits. Update your git repository after every major editing session to make sure that you don't lose your work.

Instructions

Assignment 4 consists of three (3) separate Python projects. The idea is to develop the famous SENG 265 Python Arts program incrementally for Part 3 of Assignment 4. All three parts are required. The first two parts are worth 20% each and Part 3 is worth 60%. Store the three different Python projects in **three subdirectories called a41, a42, a43 of the a4 directory**.

This assignment should be a lot of fun. Show your generated art to your family and friends.

Concentrate on learning object-oriented programming in Python, including the concepts of **classes, objects, properties, instance variables, instance methods, class variables and class methods,** and **single inheritance**. Experienced Python developers write ***Pythonic code*** which is a core skill for Python developers. In this assignment, you will level up that skill using **namedtuple** and **NamedTuple**. With the factory method **namedtuple(),** you can create immutable sequence types that allow you to access their values using descriptive field names and the **dot notation** instead of unclear integer indices. With **NamedTuple** you can create subclasses that inherit **namedtuple** properties.

Core components of this assignment include ***incremental development*** and ***Git version control.*** These software engineering skills are critical for substantial software projects and distributed development in teams.

Part 1

For Part 1 develop an ***object-oriented*** **Python project** to generate a simple HTML-SVG document (i.e., a web page) as depicted in Figure 1 below. The generated file can be viewed using your favorite web browser by rendering the SVG drawing depicted in Figure 2 below.

A starter program is provided for Part 1. However, this program is not written in an object-oriented style. So, a first step is to decompose the problem into classes including attributes (i.e., instance variables and methods). Please note that there are many different solutions to writing Part 1 in object-oriented style. The following class decomposition is one viable approach. Practice incremental development from the start and use Git for managing the incremental development.

1. Develop a Python class called **HtmlDocument** to generate an HTML page consisting of a header and a body as shown in Fig. 1 below.
2. Develop a Python **super** class called **HtmlComponent** to render an HTML component. Then use single inheritance to inherit the properties from **HtmlComponent**.
3. Develop a Python class called **SvgCanvas** to generate the **<svg>, </svg>** tags for the SVG drawing in an SVG canvas or viewport (cf. Fig. 1). This class should include a method called **gen_art** to generate the art (i.e., circles) in the SVG canvas. This functionality also be part of the **HtmlDocument** class.
4. Develop Python classes called **CircleShape, RectangleShape, EllipseShape** to draw circles, rectangles, and ellipses using the SVG **<circle>**, **<rect>** and **<ellipse>** tags, respectively, as depicted in Fig. 1 (only circles are depicted in this figure). Thus, these classes include circle, rectangle, and ellipse instance attributes and methods to draw these SVG shapes. The **CircleShape** class is sufficient for Part 1. However, the **Circle** and **RectangleShape** classes are required for Part 3. The **EllipseShape** class is optional for Part 3.
5. Write the generated HTML and SVG code to a **text file** (i.e., not to standard output).
6. Then view the generated HTML page (e.g., **part1.html**) using your favorite web browser.
7. Use **namedtuple** and **NamedTuple** as discussed in class fo ease of readability and maintenance of your code.

```
<html>
<head>
   <title>My Art</title>
</head>
<body>
   <!--Define SVG drawing box-->
   <svg width="500" height="300">
      <circle cx="50" cy="50" r="50" fill="rgb(255, 0, 0)" fill-opacity="1.0"></circle>
      <circle cx="150" cy="50" r="50" fill="rgb(255, 0, 0)" fill-opacity="1.0"></circle>
      <circle cx="250" cy="50" r="50" fill="rgb(255, 0, 0)" fill-opacity="1.0"></circle>
      <circle cx="350" cy="50" r="50" fill="rgb(255, 0, 0)" fill-opacity="1.0"></circle>
      <circle cx="450" cy="50" r="50" fill="rgb(255, 0, 0)" fill-opacity="1.0"></circle>
      <circle cx="50" cy="250" r="50" fill="rgb(0, 0, 255)" fill-opacity="1.0"></circle>
      <circle cx="150" cy="250" r="50" fill="rgb(0, 0, 255)" fill-opacity="1.0"></circle>
      <circle cx="250" cy="250" r="50" fill="rgb(0, 0, 255)" fill-opacity="1.0"></circle>
      <circle cx="350" cy="250" r="50" fill="rgb(0, 0, 255)" fill-opacity="1.0"></circle>
      <circle cx="450" cy="250" r="50" fill="rgb(0, 0, 255)" fill-opacity="1.0"></circle>
   </svg>
</body>
</html>
```

Figure 1: Generated HTML/SVG code

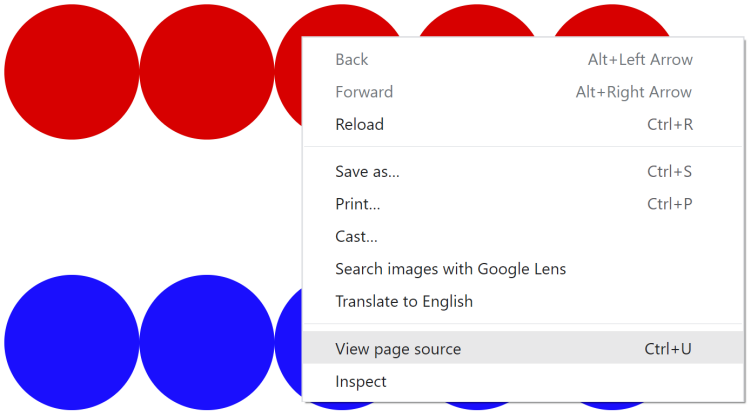


Figure 2: Sample HTML/SVG page

Part 2

Part 2 generates a table of ***random numbers*** as depicted in Table 1 below. The idea is to generate random numbers to produce random art. The hardwired numbers for the circle center, radius, and color in Fig. 1 will be replaced by random numbers in Part 3 of this assignment. Table 2 contains a set of ranges for random number generation for the different variables (e.g., radius, width, height, color, or opacity).

1. The output of Part 2 must be a table of random numbers as depicted in Table 1 below. Please note that not all columns are required for Part 3 but are required for Part 2. Also note that the random numbers you generate will of course be different than the numbers in Table 1. However, the random numbers should be within the ranges specified in Table 2 below. Use f-strings to generate each row in nicely aligned (right-justified) columns.
2. Develop a Python class called **PyArtConfig** to define ranges for the different parameters of the generated random art. For example, generate art with big or small circles and rectangles; or art with certain color shades (see images below). Use default values and keyword arguments in the **__init__** constructor of **PyArtConfig**.

Define the default ranges (e.g., minimum and maximum color values) as class variables in the **PyArtConfig** class.

3. Develop a Python class called **RandomShape** that generates a random shape according to the art style (i.e., a **PyArtConfig** object) specified. This class features three methods: **__str__(), as_Part2_line,** and **as_svg(). __str__()** returns a string of the object data nicely formatted over multiple lines. **as_Part2_line ()** returns a string of the object data in the form of a line of numbers as depicted in in Table 1. Finally, **as_svg()** returns a string of the object data in the form of SVG commands.

Table 1: Random numbers for 10 sample geometric shapes

CNT	SHA	X	Y	RAD	RX	RY	W	H	R	G	B	OP
0	2	431	13	24	27	26	18	11	82	144	73	0.1
1	1	294	28	50	23	24	32	21	135	12	62	0.2
2	1	358	264	30	30	18	32	36	179	18	77	0.9
3	2	247	179	23	27	28	22	25	9	56	37	0.7
4	2	355	263	20	17	28	21	35	19	13	137	0.8
5	1	29	217	48	25	23	29	24	64	120	54	0.8
6	2	216	290	26	22	10	37	24	173	79	20	0.9
7	0	317	150	47	18	26	28	23	50	59	161	0.5
8	2	78	53	28	13	24	30	32	176	202	153	0.3
9	2	342	205	14	12	14	18	11	153	213	233	0.5

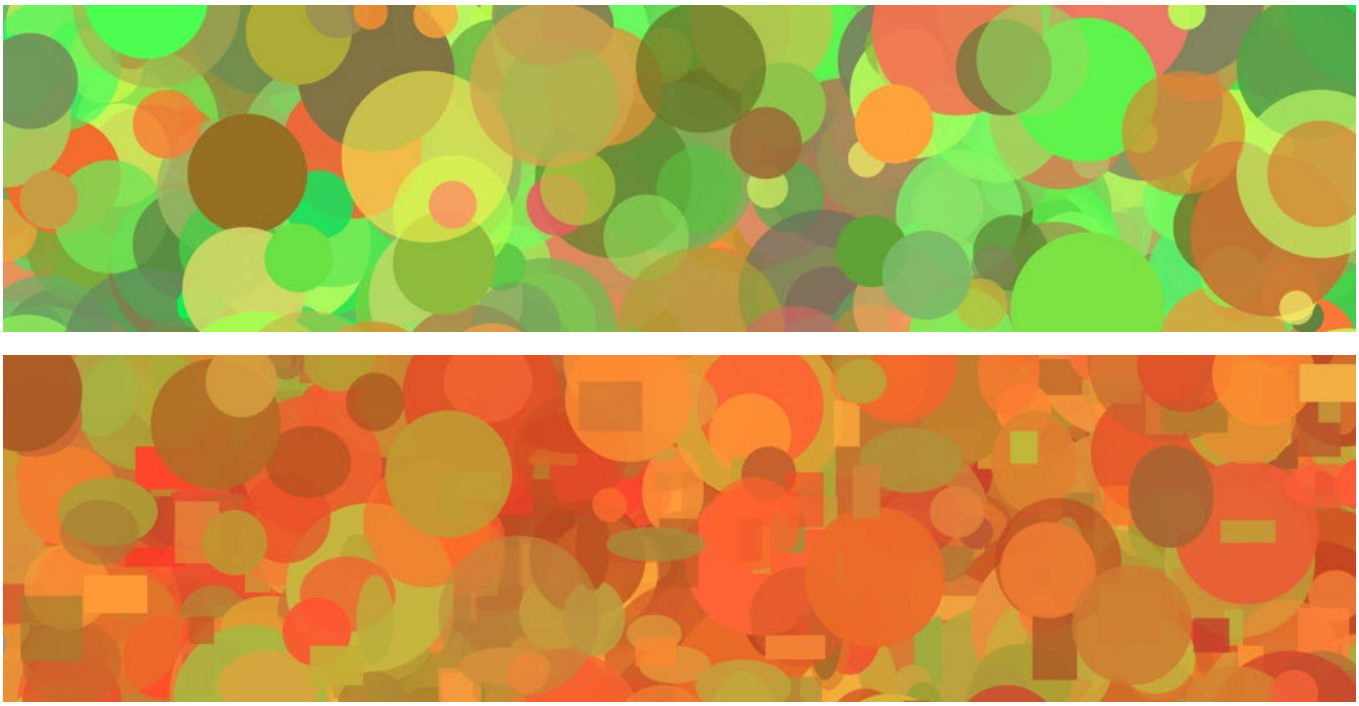
Table 2: Random numbers for 10 sample geometric shapes

VAR	Description and Range
CNT	Shape counter
SHA	Kind of shape: 0 for circle, 1 for rectangle, 3 for ellipse
X	X-coordinate of shape (e.g., circle or ellipse center, rectangle top-left corner) in viewport range
Y	Y-coordinate of shape (e.g., circle or ellipse center, rectangle top-left corner) in viewport range
RAD	Circle radius with small range 0 .. 100
RX	Ellipse radius small range 10 ..30
RY	Ellipse radius 10 ..30
W	Rectangle width small range 10 .. 100
H	Rectangle height small range 10 .. 100
R	Red color of RGB in range 0 .. 255
G	Green color of RGB in range 0 .. 255
B	Blue color of RGB in range 0 .. 255
OP	Shape opacity in range 0.0 .. 1.0

Part 3

The goal of Part 3 is to integrate the classes developed for Parts 1 and 2 into a third Python project and generate some beautiful greeting cards for your friends and family in the form of HTML-SVG pages (i.e., object of the **HtmlDocument** class). That is, Projects 1 and 2 are steppingstones for the various classes required for Part 3.

Instantiate at least **three configuration class objects to generate different art types** as depicted in Figure 3 below. Show your artistic side. Image titles and captions (e.g., postcard greetings) are optional.



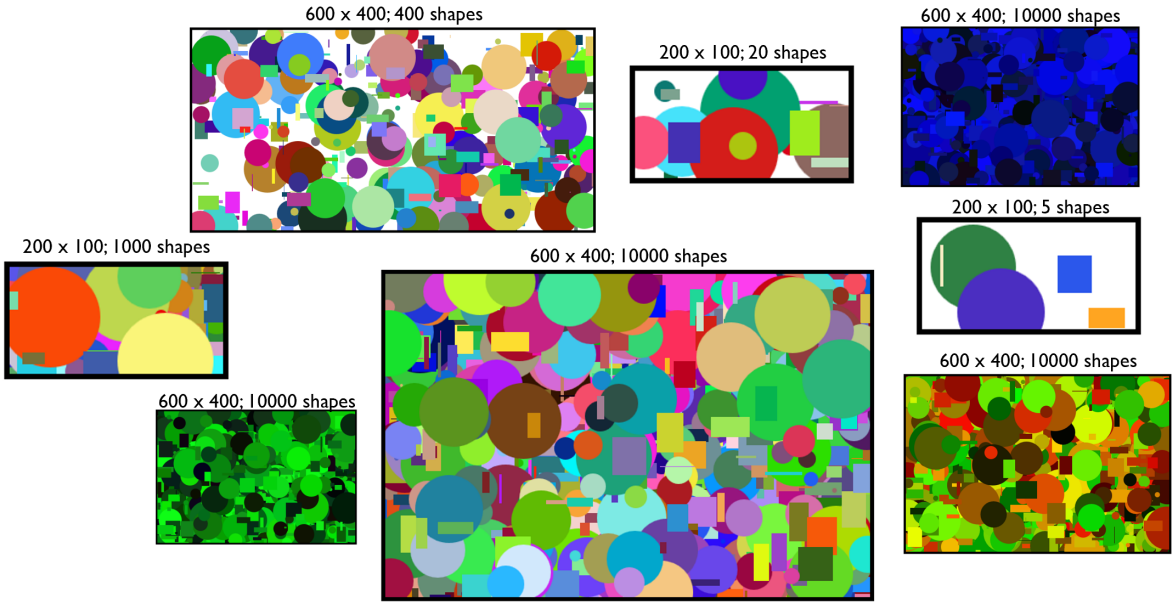


Figure 3: SENG 265 ART

Important requirements for grading

- The Python code for all three parts must be your own work and cannot be AI generated Python code.
- The most important requirement is effective object-oriented design and effective program decomposition for the three parts.
- You must use classes and objects for geometric objects (i.e., **CircleShape**, **RectangleShape**), art and shape configuration (i.e., **PyArtConfig** and **RandomShape**) as well as HTML-SVG class (i.e.,

HtmlDocument). It is helpful to model IS-A hierarchies of html components and geometric shapes using Python's single inheritance mechanism."

- To facilitate automated grading
  - the following class names are required for the required classes: **CircleShape, RectangleShape, PyArtConfig, RandomShape,** and **HtmlDocument,** and **HtmlComponent**
  - each class and each method must have a **docstring** containing the class or method name (e.g., """**CircleShape class**""" or """**drawCircle() method**""") right below the class or function header
  - all three projects must have a **main()** function and an **if __name__ == "__main__":** guard
  - for Projects 1 and 3, generate **valid, multi-line, and indented HTML-SVG files**
  - all three projects must use **Python type hints**
  - For all three projects, global, program-scope, or file/module-scope variables must not be used

What to submit

- Submit all three parts in separate directories/folders to your **a4 folder of your Git repository** as follows.
- The three different Python projects must be stored in three subdirectories/folders called a41, a42, a43 that are in your a4 directory.
- **Hint:** To verify whether you uploaded the files properly, simply clone the git repository to a new directory on your computer and check that the desired files have been placed properly.
- Part 1: Submit your Python program (a41.py) as well as the generated HTML file (a41.html)
- Part 2: Submit your Python program (a42.py) as well as a screenshot of your random table (a42.jpg)
- Part 3: Submit your Python program (a43.py) as well as three generated HTML files (a431.html, a432.html, and a433.html) and their corresponding screenshots of your art (a431.jpg, a432.jpg, and a433.jpg).

Grading assessment

- The first two parts are worth 20% each and Part 3 is worth 60%.

Additional Criteria for Qualitative Assessment

- **Documentation and commenting:** the purpose of documentation and commenting is to write information so that anyone other than yourself (with knowledge of coding) can review your program and quickly understand how it works. In terms of marking, documentation is not a large mark, but it will be part of the quality assessment.
- **Proper naming conventions:** You must use proper names for functions and variables (i.e., **PEP 008**). Using random or single character variables is considered improper coding and significantly reduces code readability. Single character variables as loop variables is fine. Please note knowledge of PEP 008 is expected for the final exam of this course.
- **Proper indentation for generated HTML/SVG code:** the generated HTML/SVG code must be properly indented according to the nesting of <html> and <svg> tags.
- **Debugging/Comment artifacts:** You must submit a clean file with no residual commented lines of code or unintended text.
- **Quality of solution:** marker will access the submission for logical and functional quality of the solution. Some examples that would result in a reduction of marks.