5.1P: In Person Check-in 2 – Answer Sheet

1. What was the most challenging aspect of the drawing tasks? Why?

   The most challenging part of the drawing tasks was implementing the `isAt` method to accurately detect mouse clicks for each shape. Rectangles used simple range checks, circles required distance calculations, and lines were especially difficult because their minimal width by clicking them with a mouse is challenging, precise point-on-line detection with SplashKit's `PointOnLine` function.

2. Review your answer to question 3 from check-in 1. Did you use any of the strategies you identified? How did they go?

   I used the strategies from my Check-in 1 answer for the ShapeDrawer project. I followed a roadmap.sh-style plan to get the big picture of SplashKitSDK and the code. I asked an AI (like ChatGPT) clear, specific questions about tricky parts like the `IsAt` method, especially for lines, which were tough due to their thin width. I also checked SplashKit's docs for details. This approach worked great—asking questions and digging deeper helped me understand the code fast and finish the assignment confidently.

3. What are some strategies for success you can start or continue using for the remainder of the semester?

   To succeed for the rest of the semester, I'll continue using a structured learning plan inspired by roadmap.sh to organize topics and prioritize key concepts. I'll keep asking targeted questions to an AI assistant, like how specific methods work or why certain approaches are used, to deepen my understanding quickly. Regularly checking official documentation, such as SplashKit's, will ensure I'm grounded in accurate details. I'll also start practicing more hands-on coding with small projects to reinforce concepts, especially for tricky tasks like precise mouse detection in programs like ShapeDrawer. Staying curious and consistent with these strategies will keep me on track.

4. Summarize how you utilized Encapsulation, Abstraction, Inheritance, and Polymorphism in your submitted tasks.

   In ShapeDrawer, **Encapsulation** protected data using private fields (e.g., _x, _y) with public properties in Shape and subclasses. **Abstraction** was achieved via the abstract Shape class, defining methods like Draw and IsAt without implementation details. **Inheritance** allowed MyRectangle, MyCircle, and MyLine to reuse Shape's fields and methods. **Polymorphism** enabled the Drawing class to call overridden methods like Draw uniformly across shape types, supporting varied behaviors.