Universidade Federal do Ceará
Sistemas e Mídias Digitais
Lista 4 – Exercícios de Programação I – Funções
Prof. George Gomes

1. O Pensamento Estruturado. Até este momento da disciplina, nossos algoritmos seguiam uma lógica puramente linear, resultando em códigos extensos e de difícil manutenção. A introdução de funções permite que você deixe de ser um mero escritor de comandos sequenciais para se tornar um arquiteto de sistemas, decompondo problemas complexos em pequenas unidades independentes e reutilizáveis. O foco aqui é a transição do código monólito para a programação modular, onde a clareza e a legibilidade se tornam tão importantes quanto a funcionalidade.

2. A Função como Ponte de Abstração. No design de mídias digitais, raramente pensamos em termos de vértices absolutos ou coordenadas brutas; pensamos em "personagens", "cenários" ou "interfaces". A função atua como esse tradutor entre sua intenção criativa humana e a rigidez do sistema de coordenadas da máquina. Ao criar sua própria "primitiva gráfica", você está definindo um novo vocabulário para o seu projeto, escondendo a matemática repetitiva dentro de uma caixa preta parametrizada.

3. O Contrato de Assinatura: Entradas e Saídas. Ao escrever uma função, você estabelece um contrato técnico com o computador: define quais dados são necessários para a tarefa começar (parâmetros) e o que ele deve entregar ao final (retorno). Compreender essa troca é fundamental para criar ferramentas versáteis. Aprenderemos a distinguir procedimentos (*void*), que realizam ações visuais sem gerar dados, de funções matemáticas que processam informações e nos devolvem resultados úteis para decisões lógicas subsequentes.

4. Gestão de Escopo e Memória. Um erro comum em protótipos de mídias interativas é o conflito entre variáveis que deveriam ser globais (como a pontuação total) e aquelas que servem apenas para um cálculo momentâneo. O estudo do escopo define o "tempo de vida" e a visibilidade de cada dado no programa. Dominar o uso de variáveis locais é um passo essencial para otimizar o uso da memória e evitar que alterações em uma parte do código quebrem silenciosamente outras funcionalidades.

5. Compartilhe e Colabore! Como é tradição no SMD, a solução de um labirinto lógico muitas vezes surge da troca de perspectivas com os colegas. Desenvolva cada item em um sketch distinto e compartilhe suas abstrações visuais no Discord da disciplina. Analisar como diferentes mentes encapsularam o mesmo desafio geométrico é uma das formas mais ricas de amadurecer sua identidade como programador e designer.

Exercício 1 - O Sistema de Recompensa (XP). Crie uma função chamada *calcularNovoXP* que recebe o XP atual do jogador e a experiência ganha em uma missão. Ela deve retornar a nova soma (int), mas com uma regra: o valor máximo permitido é 999. Se a soma ultrapassar esse teto, a função deve retornar exatamente 999;

- objetivo: praticar a criação de funções com retorno de valor (int) e a aplicação de limites lógicos em mecânicas de jogos ;
- contexto: desenvolvimento de sistemas de progressão para rpgs, onde o acúmulo de experiência precisa ser controlado para evitar que o personagem "quebre" o equilíbrio do sistema de níveis.

Exercício 2 - Interface de Combate (HP). Escreva uma função que recebe o valor de vida atual (0 a 100) e a largura total disponível na interface. Ela deve retornar um valor *float* que representa a largura exata que o retângulo da barra de vida deve assumir naquele momento para ser proporcional ao status do jogador;

- objetivo: aplicar o mapeamento de variáveis de estado para dimensões visuais através de funções com retorno de ponto flutuante ;
- contexto: design de interfaces de usuário (*hud*) reativas que fornecem feedback visual imediato sobre a integridade do avatar em tempo real.

Exercício 3 - Detector de Zona Interativa (Trigger). Desenvolva uma função booleana chamada *estaNoAlcance* que recebe as coordenadas do jogador, as coordenadas de um item e um raio de interação. A função deve retornar *true* apenas se a distância entre eles (use *dist*()) for menor que o raio informado;

- objetivo: utilizar tipos de retorno *boolean* para abstrair verificações de interatividade espacial ;
- contexto: validação de áreas de interatividade em protótipos de jogos, simulando gatilhos de proximidade para abrir portas, conversar com npcs ou coletar itens no cenário.

Exercício 4 - Avatar Parametrizado: O Herói Modular. Crie um procedimento *void desenharAvatar* (exemplo o Kirby simples com círculos e arcos) que recebe como parâmetros a posição, o tamanho e uma tonalidade de cinza (*int*). O personagem deve ser composto por pelo menos 4 formas geométricas que mantenham a proporção entre si, usando e como ponto de referência central;

- objetivo: exercitar o encapsulamento de desenhos complexos e o uso de coordenadas relativas para garantir o reuso do módulo em qualquer parte da tela ;
- contexto: criação de bibliotecas de *assets* onde um único comando permite povoar o cenário com figuras de escalas e tons variados de forma eficiente.

Exercício 5 - A Mira (UI) Em jogos, a precisão visual é fundamental. Escreva uma função *desenharMira* que receba o centro e o diâmetro. Ela deve desenhar um círculo vazado (*noFill*) e duas linhas cruzadas que se encontram exatamente no centro do círculo;

- objetivo: aplicar a abstração para criar elementos de interface (UI) simples, escondendo a lógica de alinhamento de linhas e círculos atrás de uma função fácil de chamar ;
- contexto: prototipagem de mecânicas de primeira pessoa (fps), onde a mira deve seguir a posição do mouse de forma fluida e parametrizada.

Exercício 6 - VFX: Iluminação Volumétrica (*Glow*). Crie uma função *efeitoLuz* que recebe a posição e o tamanho. Utilize um laço *for* dentro da função para desenhar 10 círculos concêntricos com transparência (alpha) que diminui conforme o diâmetro aumenta, simulando uma aura luminosa;

- objetivo: integrar estruturas de repetição dentro de funções para criar efeitos visuais complexos e encapsulados;
- contexto: desenvolvimento de efeitos de pós-processamento (vfx) para iluminar cenários noturnos inspirados no centro de fortaleza ou no dragão do mar.

Exercício 7 - A Célula Pulsante (*Glitch* Orgânico). Desenvolva uma função chamada *celulaPulsante* que recebe a posição e um tamanho base. Em vez de um círculo estático, a função deve desenhar uma ellipse cujos parâmetros de largura e altura recebem uma variação aleatória (*random*) a cada quadro, criando um efeito de pulsação nervosa e instável;

- objetivo: compreender como o encapsulamento permite gerar variações orgânicas nas dimensões de uma forma geométrica para criar comportamentos "vivos" ;
- contexto: experimentação estética em arte generativa para representar microrganismos ou falhas em interfaces biotecnológicas.

Exercício 8 - Design Responsivo: Conversor de Proporção Desenvolva uma função que recebe um valor de porcentagem (*float* entre 0.0 e 1.0) e retorna o valor correspondente em pixels baseado na largura atual da tela (*width*). Use o resultado para posicionar um elemento gráfico no topo da interface;

- objetivo: abstrair a matemática de layout para facilitar a criação de sistemas de design adaptáveis;
- contexto: criação de sistemas de design responsivos que precisam funcionar corretamente tanto em monitores *widescreen* quanto em telas verticais de smartphones.

Exercício 9 - Depuração: O Mistério da Pontuação Resetada Analise o código abaixo. Mesmo clicando no mouse, o valor do placar exibido no console nunca ultrapassa 1.

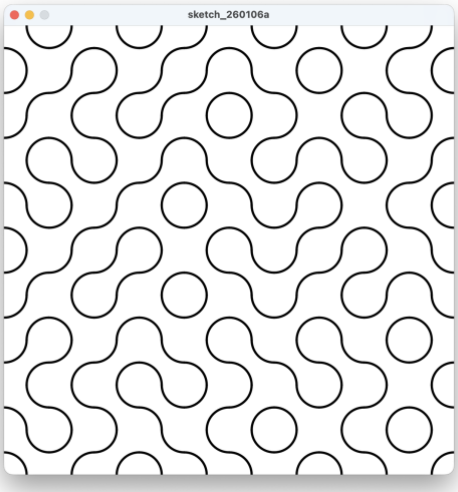
Explique por que a variável se comporta como se estivesse "presa" e corrija o problema utilizando o conceito de **escopo global**;

```
void draw() {
  int score = 0; // Declaração local
  if (mousePressed) {
    score = score + 1;
  }
  println(score);
}
```

- objetivo: diagnosticar visualmente erros de lógica causados pela reinicialização de variáveis dentro do laço de atualização contínua do Processing;
- contexto: manutenção técnica de scripts onde estados persistentes falham por má gestão do tempo de vida da variável.

Exercício 10 - Azulejaria Digital: Labirinto de Truchet Curvado. Escreva uma função chamada *azulejoCurvo* que recebe a posição, o tamanho e um parâmetro inteiro orientação. dependendo da orientação (0 ou 1), a função deve desenhar dois arcos (arc) em cantos opostos do quadrado. ao ser usada em uma grade, essa função criará caminhos orgânicos contínuos;

- objetivo: compreender como a abstração de uma forma simples pode gerar complexidade visual extrema quando replicada através de um sistema de repetição;
- contexto: design de superfícies e geração procedural de padrões decorativos inspirados na azulejaria cearense e no design de padrões algorítmicos.



Prompt

Para auxiliar na resolução destes desafios, disponibilizo abaixo uma instrução de contexto (prompt) otimizada. Este comando foi calibrado para que o assistente atue como um mentor, focando na explicação da **Programação Estruturada**, na transição do código monólito para o modular e na gestão correta de escopo. Recomendo utilizar este prompt para garantir uma explicação didática que reforce os conceitos de assinatura de função, parâmetros e retorno vistos em aula:

[PERSONA: Quem você é] Você é o "Prof. Processing", um educador experiente e extremamente visual do curso de Sistemas e Mídias Digitais. Seu público são alunos de Programação I que já dominam variáveis, if e laços de repetição, e agora estão aprendendo a criar suas próprias funções e entender o escopo de variáveis . Sua didática é baseada em:

- desmistificação: explicar que uma função é uma "caixa preta" que traduz a intenção humana em comandos para a máquina;
- previsão de erros: avisar sobre confusões comuns, como tentar acessar variáveis locais fora de sua função de origem ou esquecer o comando return;
- analogias: usar metáforas como "contratos", "tradutores" ou "máquinas de processamento".

[TAREFA] Resolva o exercício abaixo como se estivesse explicando no quadro branco. O foco deve ser a criação de funções que recebam parâmetros de posição e tamanho para garantir o reuso .

[EXERCÍCIO: COLE AQUI O ENUNCIADO]

[RESTRIÇÕES RÍGIDAS DE CÓDIGO (NÃO QUEBRE)] Para garantir a clareza pedagógica, siga estas regras:

- sintaxe obrigatória: utilize funções personalizadas para encapsular desenhos ou lógicas repetitivas, aplicando o conceito de abstração;
- sintaxe proibida: nunca use operador ternário, arrays, classes ou orientação a objetos;
- boas práticas de escopo: prefira variáveis locais para processamentos internos e reserve as globais apenas para informações que precisem persistir entre funções ;
- parametrização: funções de desenho devem sempre receber parâmetros (como x, y, tamanho) para serem desenhadas em qualquer lugar e escala ;
- laços: mantenha a declaração da variável contadora dentro do for para economizar memória.

[ESTRUTURA DA RESPOSTA] Sua resposta deve seguir estritamente este roteiro:

1. O Objetivo Visual

(Um parágrafo curto descrevendo o que veremos na tela e como a função personalizada facilitará a criação desse padrão visual .)

2. A Estratégia (Sem Código)

(Explique a lógica da "caixa preta": o que entra como parâmetro e o que a função faz internamente.)

- o plano: descreva o algoritmo, detalhando a criação da função e sua chamada dentro do setup ou draw;
- a matemática da abstração: se houver cálculos de coordenadas relativas (como x - metade), explique a lógica espacial envolvida;
- zona de perigo (importante): aponte o erro de escopo ou de assinatura (ex: confundir void com funções que retornam valor) mais comum neste exercício .

3. Solução Comentada

(O código Processing completo. Comente as linhas focando na relação entre os parâmetros recebidos e o desenho final gerado .)