# Final Exam Review

## CS 105 • Fall 2024

Jackson Eshbaugh

Lafayette College

2024-12-19

# Outline

Variables & Functions

Control Flow

Object Oriented Programming

# Outline

Variables & Functions

Control Flow

Object Oriented Programming

# Variables

## Types of Variables

- In Java (Processing), each variable has a type that describes the data it holds.

# Variables

## Types of Variables

- In Java (Processing), each variable has a type that describes the data it holds.
- Examples include:
  - `int`
  - `float`
  - `String`
  - `char`

# Variables

## Types of Variables

- In Java (Processing), each variable has a type that describes the data it holds.
- Examples include:
  - `int`
  - `float`
  - `String`
  - `char`
- Any class you create is also a variable type. For example, `String` is a class in Java.

# Variables

## Declaring & Initializing Variables

- **Declaring** a variable is the act of allocating space for it in memory. This is like creating an empty box in memory that might be filled later.

# Variables

## Declaring & Initializing Variables

- **Declaring** a variable is the act of allocating space for it in memory. This is like creating an empty box in memory that might be filled later.
- **Initializing** a variable is the act of filling this box.

# Variables

## Declaring & Initializing Variables

- **Declaring** a variable is the act of allocating space for it in memory. This is like creating an empty box in memory that might be filled later.
- **Initializing** a variable is the act of filling this box.

**Remark**

Declaring and initializing can be combined into one step: `int i = 12;` is equivalent to

```
int i;
i = 12;
```

# Variables

## Variable Scope

- Variables exist in a context. We refer to this context as **scope**.

# Variables

## Variable Scope

- Variables exist in a context. We refer to this context as **scope**.
- Scope describes where a variable is accessible.

# Variables

## Variable Scope

- Variables exist in a context. We refer to this context as **scope**.
- Scope describes where a variable is accessible.
- For example, if a variable is declared within a function, it cannot be accessed outside of that function.

# Variables

## Variable Scope

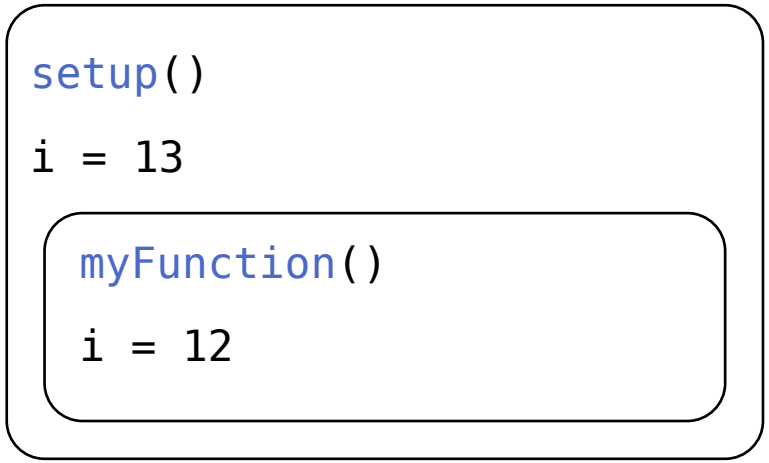
- For example, if a variable is declared within a function, it cannot be accessed outside of that function.

### Example Code

```
void setup() {
  int i = 13;
  println(i); // Result: 13
}

void myFunction() {
  int i = 12;
}
```

### Scope Diagram



# Functions

- A **function** is a *reusable* block of code that performs some task and optionally, *returns* something.

# Functions

- A **function** is a *reusable* block of code that performs some task and optionally, *returns* something.
- To declare a function:

# Functions

- A **function** is a *reusable* block of code that performs some task and optionally, *returns* something.
- To declare a function:

1. Function Header

# Functions

- A **function** is a *reusable* block of code that performs some task and optionally, *returns* something.
- To declare a function:

1. Function Header
2. Algorithm

# Functions

- A **function** is a *reusable* block of code that performs some task and optionally, *returns* something.
- To declare a function:

1. Function Header
2. Algorithm
3. Implementation

# Functions

## Writing A Function: Example

Write a function that raises some integer $x$ to the power of another integer $y$. In other words, this function computes & returns the value $x^y$.

# Functions

## Writing A Function: Example

Write a function that raises some integer $x$ to the power of another integer $y$. In other words, this function computes & returns the value $x^y$.

1. Function Header

# Functions

## Writing A Function: Example

Write a function that raises some integer $x$ to the power of another integer $y$. In other words, this function computes & returns the value $x^y$.

1. Function Header

```
int pow(int x, int y)
```

# Functions

## Writing A Function: Example

Write a function that raises some integer $x$ to the power of another integer $y$. In other words, this function computes & returns the value $x^y$.

1. Function Header

```
int pow(int x, int y)
```

2. Algorithm:

# Functions

## Writing A Function: Example

Write a function that raises some integer $x$ to the power of another integer $y$. In other words, this function computes & returns the value $x^y$.

1. Function Header

```
int pow(int x, int y)
```

2. Algorithm:

Multiply $x$ by itself $y$ times.

# Functions

## Writing A Function: Example

Write a function that raises some integer $x$ to the power of another integer $y$. In other words, this function computes & returns the value $x^y$.

3. Implementation

# Functions

## Writing A Function: Example

Write a function that raises some integer $x$ to the power of another integer $y$. In other words, this function computes & returns the value $x^y$.

3. Implementation

```
int pow(int x, int y) {

}
```

# Functions

## Writing A Function: Example

Write a function that raises some integer $x$ to the power of another integer $y$. In other words, this function computes & returns the value $x^y$.

3. Implementation

```
int pow(int x, int y) {

  while(y > 0) {

  }
}
```

# Functions

## Writing A Function: Example

Write a function that raises some integer $x$ to the power of another integer $y$. In other words, this function computes & returns the value $x^y$.

3. Implementation

```
int pow(int x, int y) {

  while(y > 0) {
    x *= x;
  }
}
```

# Functions

## Writing A Function: Example

Write a function that raises some integer $x$ to the power of another integer $y$. In other words, this function computes & returns the value $x^y$.

3. Implementation

```
int pow(int x, int y) {

  while(y > 0) {
    x *= x;
  }
  return x;
}
```

# Functions

## Function Overloading

- Function overloading allows multiple functions to have the same name, but different parameters.

# Functions

## Function Overloading

- Function overloading allows multiple functions to have the same name, but different parameters.
- Example:

```
int add(int a, int b) {
  return a + b;
}
```

# Functions

## Function Overloading

- Function overloading allows multiple functions to have the same name, but different parameters.
- Example:

```
int add(int a, int b) {
  return a + b;
}

double add(double a, double b) {
  return a + b;
}
```

# Functions

## Function Overloading

- Function overloading allows multiple functions to have the same name, but different parameters.
- Example:

```
int add(int a, int b) {
  return a + b;
}

double add(double a, double b)
{
  return a + b;
}
```

**Remark**

Overloading is commonplace in Java and Processing.

# Functions

## Function Overloading

- Function overloading allows multiple functions to have the same name, but different parameters.
- Example:

```
int add(int a, int b) {
  return a + b;
}

double add(double a, double b)
{
  return a + b;
}
```

**Warning**

Note that overloading *is not* the same concept as overriding.

# Functions

## Testing Functions

- Testing is crucial to ensure that functions work as expected.

# Functions

## Testing Functions

- Testing is crucial to ensure that functions work as expected.

1. Ground truth table

# Functions

## Testing Functions

- Testing is crucial to ensure that functions work as expected.

1. Ground truth table
2. Automated testing

# Functions

## Testing Functions

Write a test for the function below:

```
int pow(int x, int y) {
  while(y > 0) {
    x *= x;
  }
  return x;
}
```

# Functions

## Testing Functions

Write a test for the function below:

```
int pow(int x, int y) {
  while(y > 0) {
    x *= x;
  }
  return x;
}
```

- Ground truth table:

| Input | Expected Output |
|---|---|
| 2, 3 | 8 |
| 3, 2 | 9 |
| 4, 0 | 1 |
| 5, 1 | 5 |

# Functions

## Testing Functions

Write a test for the function below:

```
int pow(int x, int y) {
  while(y > 0) {
    x *= x;
  }
  return x;
}
```

### Automated Testing

```
void testPow() {
  int in1 = 2;
  int in2 = 3;
  int expected = 8;
  int actual = pow(in1, in2);
  if(actual == expected) {
    println("test passed");
  } else {
    println("test failed");
  }
}
```

# Processing Functions & Variables

## Processing Functions

- Processing provides several built-in functions to handle events and drawing.

# Processing Functions & Variables

## Processing Functions

- Processing provides several built-in functions to handle events and drawing.
- `void setup()`: Called once when the program starts.

```
void setup() {
  size(400, 400);
}
```

# Processing Functions & Variables

## Processing Functions

- Processing provides several built-in functions to handle events and drawing.
- `void setup()`: Called once when the program starts.

```
void setup() {
  size(400, 400);
}
```

- `void draw()`: Continuously executes the lines of code contained inside its block until the program is stopped.

```
void draw() {
  background(255);
  ellipse(mouseX, mouseY, 50, 50);
}
```

# Processing Functions & Variables

- `void mousePressed()`: Called once after every time a mouse button is pressed.

```
void mousePressed() {
  println("Mouse pressed at: " + mouseX + ", " + mouseY);
}
```

# Processing Functions & Variables

- `void mousePressed()`: Called once after every time a mouse button is pressed.

```
void mousePressed() {
  println("Mouse pressed at: " + mouseX + ", " + mouseY);
}
```

- `void keyPressed()`: Called once every time a key is pressed.

```
void keyPressed() {
  println("Key pressed: " + key);
}
```

# Processing Functions & Variables

## Processing Variables

- Processing provides several built-in variables to get information about the environment.

# Processing Functions & Variables

## Processing Variables

- Processing provides several built-in variables to get information about the environment.
- `height`: System variable that stores the height of the display window.
- `width`: System variable that stores the width of the display window.
- `key`: System variable that stores the value of the last key pressed.

```
void draw() {
  if (key == 'a') {
    println("The 'a' key was pressed.");
  }
}
```

# Outline

Variables & Functions

Control Flow

Object Oriented Programming

# Control Flow

## If Statements

- If statements allow you to execute code conditionally.

# Control Flow

## If Statements

- If statements allow you to execute code conditionally.
- Example:

```
if (x > 0) {
  println("x is positive");
} else {
  println("x is not positive");
}
```

# Control Flow

## Loops

- Loops allow you to execute code repeatedly.

# Control Flow

## Loops

- Loops allow you to execute code repeatedly.
- Example:

```
for (int i = 0; i < 10; i++) {
  println(i);
}
```

# Outline

Variables & Functions

Control Flow

Object Oriented Programming

# Object Oriented Programming

## Classes & Objects

- A class is a blueprint for creating objects.

# Object Oriented Programming

## Classes & Objects

- A class is a blueprint for creating objects.
- Example:

```
class Dog {
  String name;
  int age;

  void bark() {
    println("Woof!");
  }
}
```

# Object Oriented Programming

## Inheritance

- Inheritance allows a class to inherit properties and methods from another class.

# Object Oriented Programming

## Inheritance

- Inheritance allows a class to inherit properties and methods from another class.
- Example:

```
class Animal {
  void eat() {
    println("This animal is eating");
  }
}

class Dog extends Animal {
  void bark() {
    println("Woof!");
  }
}
```