WELCOME TO
CSCE 221: DATA STRUCTURES
SYLLABUS
CH. 1
JAVA PRIMER
WHAT IS COMPUTER SCIENCE?
• Study of algorithms
• Study of computing tools

• It is not just:
  • Programming
  • Electronics
  • Etc.

• In this class, we formalize this study of algorithms through the basics of data structures – a bread-and-butter component of almost all algorithms.
Even though computer science is not about the computer, we still need to tell the computer what to do!

We do this through programming, or the act of writing a computer program, known as software – it's just instructions to the computer.

We will use Java.

NOTE – This is an arbitrary choice. All languages build on the same basic building blocks. So Java is merely the vessel to our exploration of data structures!
WHY JAVA?

• Every language has its own advantages and disadvantages
• Java
  • Widely used and available
  • Great for developing large programs
  • Embraces full set of modern abstractions
  • Variety of automatic checks for mistakes in programs
• Our study will
  • Develop general programming skills that are applicable to many languages

“There are only two kinds of programming languages: those people always gripe about and those that nobody use.”
– Paraphrased from Bjarne Stroustrup
ALGORITHMS

- In this class, we focus on honing the tools to express and model algorithms and software. We will learn not only Java, but something called Pseudocode.
- **Pseudocode** is a detailed and stylized description for program and algorithm design. Often more natural than true language.
  - Look at my examples in the homeworks and LaTeX tips

- **Code**
  - Sometimes hard to read
  - Can only use a restricted subset of math and natural language
  - Generates compile errors if done improperly
  - Runs on a computer

- **Pseudocode**
  - Stylized and easy to read
  - Can use math and natural language
  - Is not compiled, therefore ‘;’, ‘{‘, etc “don’t matter”
  - Does not run on a computer
ALGORITHM BASICS

PROGRAM STRUCTURE, VARIABLES, AND ARRAYS IN PSEUDOCODE, PYTHON, AND JAVA
PROGRAMMING PROCESS

Using Pseudocode

Write Program

Mentally Trace

Output

Using an interpreter, e.g., Python

Write Program

Interpreter
  • Reads language directly

Execute Program

Output

Using a compiler, e.g., Java

Write Program

Compiler (e.g., javac)
  • Translation to another language

Machine Code
  • Specific for an architecture

Execute Program (e.g., java)

Output

Using an interpreter, e.g., Python

Write Program

Interpreter
  • Reads language directly

Execute Program

Output

Using a compiler, e.g., Java

Write Program

Compiler (e.g., javac)
  • Translation to another language

Machine Code
  • Specific for an architecture

Execute Program (e.g., java)

Output

Using an interpreter, e.g., Python

Write Program

Interpreter
  • Reads language directly

Execute Program

Output

Using a compiler, e.g., Java

Write Program

Compiler (e.g., javac)
  • Translation to another language

Machine Code
  • Specific for an architecture

Execute Program (e.g., java)

Output
A SIMPLE PROGRAM
HELLO WORLD

PSEUDOCODE

Algorithm HelloWorld
1. Print("Hello World!") {Hello World!}

PYTHON

HelloWorld.py
1. # Hello World!
2. print("Hello World")

JAVA CODE

HelloWorld.java
1. // Hello World!
2. public class HelloWorld {
3.     public static void main(String[] args) {
4.         System.out.println("Hello World!");
5.     }
6. }
A SIMPLE PROGRAM
HELLO WORLD

HelloWorld.java

1. // Hello World!
2. public class HelloWorld {
3.   public static void main(String[] args) {
4.     System.out.println("Hello World!");
5.   }
6. }

Public means anyone can see it
All Java code must belong to a class
Curly braces denote scope
Method belongs to the class
Return type of the method
Name of the method
Formal parameters of the method
Method to print
Actual parameters passed
Statements end with a semicolon
COMPILER RECOMMENDATION

• Personally, I prefer a simple text editor, e.g., Visual Studio Code or Sublime and the terminal or command prompt to compile
  • javac command will compile and java command will run program

• IDE, Integrated Development Environments, combine compilation with execution
  • Any of BlueJay, Dr. Java, Netbeans, Eclipse, etc. will work just fine

• Up to you to learn one how to use it

• I recommend trying many and picking your favorite
PROGRAMMING ERRORS

• Syntax Errors
  • Detected by the compiler

• Runtime Errors also called Exceptions
  • Causes the program to abort

• Logic Errors
  • Produces incorrect result
  • Must use tracing to help correct
SYNTAX ERRORS

• **Syntax errors** are errors from incorrectly written Java code. The compiler (**javac**) tells you these.

• **Anatomy of a compiler error:**
  filename.java:line_num: error: Confusing description of error including code where it occurs.

• **Deal with by experience, google, stack overflow, etc. After you have exhausted these resources...piazza/ask me. Advice, always handle the first error...not the last one.**

```java
// This program prints Hello World!
public Class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

Can anyone spot the syntax errors?
RUNTIME ERRORS

- **Runtime errors** occur from impossible commands encountered while executing the program (with java)
- Deal with by tracing and debugging

1. // This program prints Hello World!
2. public class HelloWorld {
3.  public static void main(String[] args) {
4.    System.out.println(1/0);
5.  }
6. }

Can anyone spot the runtime error?
LOGIC ERRORS

- **Logic errors** are incorrect computations that affect the correctness of your program
- Deal with by tracing and debugging

1. // This program prints Hello World!
2. public class HelloWorld {
3.    public static void main(String[] args) {
4.        System.out.println("Celsius 35 is Fahrenheit degree ");
5.        System.out.println((9 / 5) * 35 + 32);
6.    }
7.}

Can anyone spot the logic error?
VARIABLES AND EXPRESSIONS

PSEUDOCODE
• Variable declarations optionally include the type (preferred), assignment is done with '←' (pronounced 'gets'), and expressions are mathematic formulas

Integer \( r \leftarrow 10 \)
Real \( a \leftarrow \pi r^2 \)

JAVA CODE
• Variable declarations require the type, assignment is done with '='; and expressions use operators

```java
int r = 10;
double a = Math.PI * r * r;
```

PYTHON
• Variable declarations optionally include the type (not required), assignment is done with '='; and expressions use operators

```python
import math
r = 10
a = math.pi * r * r
```
IDENTIFIERS, LITERALS, AND CONSTANTS

• An **identifier** is an arbitrarily long sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($)
  • Identifiers name classes, methods, and variables
  • An identifier cannot start with a digit or be a reserved keyword

• A **literal** is a constant value that appears directly in the program

• A constant, a variable whose value does not change, is denoted as **final**
  • **final double** pi = 3.14159;
**PRIMITIVE TYPES**

- **Primitive types** are those that map directly into the bits of memory (can be represented directly as 0s and 1s)
- Range of data and operations are defined for the various types

<table>
<thead>
<tr>
<th>Pseudocode</th>
<th>Java</th>
<th>Storage in Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>boolean</td>
<td>True or false value</td>
</tr>
<tr>
<td>Character</td>
<td>char</td>
<td>16 bit Unicode character</td>
</tr>
<tr>
<td>Integer</td>
<td>byte, short, int, or long</td>
<td>8, 16, 32, or 64 bit signed integer</td>
</tr>
<tr>
<td>Real</td>
<td>float or double</td>
<td>32 or 64 bit floating point number (IEEE 754)</td>
</tr>
</tbody>
</table>
### NUMERIC OPERATORS

<table>
<thead>
<tr>
<th>Name</th>
<th>Pseudocode</th>
<th>Python</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>$a + b$</td>
<td>a + b</td>
<td>a + b</td>
</tr>
<tr>
<td>Subtraction</td>
<td>$a - b$</td>
<td>a - b</td>
<td>a - b</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$ab$</td>
<td>a * b</td>
<td>a * b</td>
</tr>
<tr>
<td>Division</td>
<td>$\frac{a}{b}$</td>
<td>a/b #Floating point division a//b #Integer division</td>
<td>a/b //Depends on type</td>
</tr>
<tr>
<td>Remainder</td>
<td>$a \mod b$</td>
<td>a % b</td>
<td>a % b</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>$a^b$</td>
<td>a ** b</td>
<td>No such operator</td>
</tr>
</tbody>
</table>
AUGMENTED ASSIGNMENT, INCREMENT, AND DECREMENT

- These do not exist in pseudocode, but can make programming simpler for expressions like \( a = a \ <\text{op}> \ b \)

<table>
<thead>
<tr>
<th>Name</th>
<th>Python</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>( a += b )</td>
<td>( a += b )</td>
</tr>
<tr>
<td>Subtraction</td>
<td>( a -= b )</td>
<td>( a -= b )</td>
</tr>
<tr>
<td>Multiplication</td>
<td>( a *= b )</td>
<td>( a *= b )</td>
</tr>
<tr>
<td>Division</td>
<td>( a/=b ) #Floating point division ( a//=b ) #Integer division</td>
<td>( a/=b ) //Depends on type</td>
</tr>
<tr>
<td>Remainder</td>
<td>( a %= b )</td>
<td>( a %= b )</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>( a **= b )</td>
<td>No such operator</td>
</tr>
<tr>
<td>Pre and post increment</td>
<td>No such operator</td>
<td>++a or a++</td>
</tr>
<tr>
<td>Pre and post decrement</td>
<td>No such operator</td>
<td>--a or a--</td>
</tr>
</tbody>
</table>
# RELATIONAL OPERATORS

<table>
<thead>
<tr>
<th>Name</th>
<th>Pseudocode</th>
<th>Python and Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equal to</td>
<td>$a = b$</td>
<td>$a == b$</td>
</tr>
<tr>
<td>Not equal to</td>
<td>$a \neq b$</td>
<td>$a != b$</td>
</tr>
<tr>
<td>Less than</td>
<td>$a &lt; b$</td>
<td>$a &lt; b$</td>
</tr>
<tr>
<td>Less than or equal to</td>
<td>$a \leq b$</td>
<td>$a &lt;= b$</td>
</tr>
<tr>
<td>Greater than</td>
<td>$a &gt; b$</td>
<td>$a &gt; b$</td>
</tr>
<tr>
<td>Greater than or equal to</td>
<td>$a \geq b$</td>
<td>$a &gt;= b$</td>
</tr>
</tbody>
</table>
## LOGICAL OPERATORS

<table>
<thead>
<tr>
<th>Name</th>
<th>Pseudocode</th>
<th>Python</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or</td>
<td>$a \lor b$</td>
<td>a or b</td>
<td>a</td>
</tr>
<tr>
<td>And</td>
<td>$a \land b$</td>
<td>a and b</td>
<td>a &amp;&amp; b</td>
</tr>
<tr>
<td>Not</td>
<td>$\neg a$</td>
<td>not a</td>
<td>!a</td>
</tr>
<tr>
<td>Operator</td>
<td>Description</td>
<td>Associativity</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td>12 [], ., ()</td>
<td>access array element, access object member, parentheses</td>
<td>left to right</td>
<td></td>
</tr>
<tr>
<td>11 ++, --</td>
<td>unary post-increment/decrement</td>
<td>not associative</td>
<td></td>
</tr>
<tr>
<td>10 ++, --, +, -, !, ~</td>
<td>unary pre-increment/decrement, plus, minus, logical NOT</td>
<td>right to left</td>
<td></td>
</tr>
<tr>
<td>9 (, new</td>
<td>Cast, object creation</td>
<td>right to left</td>
<td></td>
</tr>
<tr>
<td>8 *, /, %</td>
<td>multiplicative</td>
<td>left to right</td>
<td></td>
</tr>
<tr>
<td>7 +, -</td>
<td>Additive, string concatenation</td>
<td>left to right</td>
<td></td>
</tr>
<tr>
<td>6 &lt;, &lt;=, &gt;, &gt;=, instanceof</td>
<td>relational</td>
<td>not associative</td>
<td></td>
</tr>
<tr>
<td>5 ==, !=</td>
<td>equality</td>
<td>left to right</td>
<td></td>
</tr>
<tr>
<td>4 &amp;&amp;</td>
<td>logical AND</td>
<td>left to right</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>logical OR</td>
</tr>
<tr>
<td>2 ?:</td>
<td>ternary</td>
<td>right to left</td>
<td></td>
</tr>
<tr>
<td>1 =, +=, -=, *=, /=, %=</td>
<td>assignment</td>
<td>right to left</td>
<td></td>
</tr>
</tbody>
</table>
TYPE CASTING

- **Implicit casting** is done when the range of the data increases, e.g., `int` to `double`. Implicit means automatic by Java
  - `double d = 3; //type widening`

- **Explicit casting** is done when the range of the data decreases, e.g., `double` to `int`. Explicit must be stated by the programmer
  - `int i = (int)3.9;` (type narrowing, fraction will be truncated not rounded!)

- **What is wrong?**
  - `int x = 5 / 2.0;`

Range increases

`byte, short, int, long, float, double`
ARRAYS

• An **array** is a sequenced collection of variables all of the same type. Each variable, or **cell**, in an array has an **index**, which uniquely refers to the value stored in that cell. The cells of an array, $A$, are numbered $0, 1, 2$, and so on.

• Each value stored in an array is often called an **element** of that array.

• Can be multi-dimensional
ARRAYS
STATIC INITIALIZATION

PSEUDOCODE
1. Array of DataType arr ← {value1, value2, ..., value3}

Example
1. Array of Integers arr ← {1, 2, 3}

PYTHON
Does not exist. Use lists to emulate.
1. arr = [value1, value2, ..., valueN]

Example
1. arr = [1, 2, 3]

JAVA CODE
1. DataType[] arr = {value1, value2, ..., valueN};
Or
1. DataType arr[] = {value1, value2, ..., valueN};

Example
1. int[] arr = {1, 2, 3};
ARRAYS
DYNAMIC INITIALIZATION

PSEUDOCODE

1. Array of DataType arr of size \( n \)

PYTHON

Does not exist. Use lists to emulate.

1. \( \text{arr} = \#[None] \ast n \)

JAVA CODE

1. \( \text{DataType}[] \text{ arr} = \text{new DataType}[n]; \)

Or

1. \( \text{DataType} \text{ arr}[] = \text{new DataType}[n]; \)
## Arrays Length

<table>
<thead>
<tr>
<th>Pseudocode</th>
<th>Python</th>
</tr>
</thead>
</table>
| 1. arr.length  
Or preferably  
1. |arr| | 1. len(arr) |

<table>
<thead>
<tr>
<th>Java Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. arr.length</td>
</tr>
</tbody>
</table>
CONTROL FLOW

CONTROL FLOW IN PSEUDOCODE, PYTHON, AND JAVA
CONTROL FLOW

• Control flow.
  • Sequence of statements that are actually executed in a program.
  • Conditionals and loops: enable us to choreograph control flow.

Notation
• Block – statement of code
• Diamond – conditional
• Open circle – start/end of algorithm

straight-line control flow

control flow with conditionals and loops
CONDITIONALS

- Conditionals create branches program execution
CONDITIONALS IN PSEUDOCODE

1. if booleanCondition then
2. {Perform some operation}
3. else
4. {Perform some other operation}

1. if booleanCondition then
2. {Perform some operation}
3. else if booleanCondition2 then
4. {Perform some other operation}
5. {...}
6. else
7. {Perform final option of operation}
CONDITIONALS
IN PYTHON

1. if booleanCondition:
2.   # Perform some operation
3. else:
4.   # Perform some other operation

1. if booleanCondition:
2.   # Perform some operation
3. elif booleanCondition2:
4.   # Perform some other operation
5.   # …
6. else:
7.   # Perform final option of operation
CONDITIONALS
IN JAVA

1. if (booleanCondition) {
2.   // Perform some operation
3. }
4. else {
5.   // Perform some other operation
6. }

1. if (booleanCondition) {
2.   // Perform some operation
3. }
4. else if (booleanCondition2) {
5.   // Perform some other operation
6. }
7. //...
8. else {
9.   // Perform final option of operation
10. }

• Note - parenthesis are required and curly braces are optional ONLY FOR single statement blocks
ADVANCED CONDITIONAL IN JAVA
SWITCH STATEMENT

• Special syntax for large if-else if blocks

• The switch-expression must yield an integral, string, or enumeration value and must always be enclosed in parentheses.

• The value, ..., and valueN must have the same data type as the value of the switch-expression and must be constants.

• The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression.

• The keyword break is optional, but it should be used at the end of each case. It jumps to the end of the switch. If the break statement is not present, the next case statement will be executed.

• The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.

• When the value in a case statement matches the value of the switch-expression, the statements starting from this case are executed until either a break statement or the end of the switch statement is reached.

```
1. switch (switch-expression) {
2.   case value1: statement(s)1;
3.     break;
4.   case value2: statement(s)2;
5.     break;
6.     // ...
7.   case valueN: statement(s)N;
8.     break;
9.   default: statement(s)-for-default;
10. }
```
ADVANCED CONDITIONALS IN JAVA

CONDITIONAL EXPRESSIONS

1. if (x > 0)
2.    y = 1;
3. else
4.    y = -1;

• Is equivalent to a special ternary operator
  y = (x > 0) ? 1 : -1;
  (boolean-expression) ? expression1 : expression2;
Loops allow repetition during program execution.
WHILE LOOPS

PSEUDOCODE
1. while booleanCondition do
2. {Perform some operations}

PYTHON
1. while booleanCondition:
2. # Perform some operations

JAVA CODE
1. while(booleanCondition) {
2. // Perform some operations
3. }
DO-WHILE LOOPS

PSEUDOCODE
1. repeat
2. {Perform some operations}
3. until booleanCondition

PYTHON
Does not exist. Must emulate with basic while loop

JAVA CODE
1. do {
2. // Perform some operations
3. } while(booleanCondition);
FOR LOOPS
BASIC

PSEUDOCODE
1. for initialization to bound
   step increment do
2. {Perform some operations}

Step is optional.

PYTHON
Does not exist must emulate with enhanced for-loop (for-each loop).

JAVA CODE
1. for initialization; booleanCondition; increment) {
2.   // Perform some operations
3. }

.Diagram showing the flow of a for loop with initialization, condition, and action steps.
FOR-EACH LOOPS
ENHANCED FOR LOOPS

PSEUDOCODE
1. for all variableDeclaration ∈ sequence do
2. {Perform some operations}

∈ means 'in'.

PYTHON
1. for variableDeclaration in sequence:
2. # Perform some operations

JAVA CODE
1. for (variableDeclaration : sequence) {
2.   // Perform some operations
3. }

[Diagram of enhanced for loop with actions labeled: initialization, condition check, statement(s), action after each iteration, and loop continuation conditions.]
COMPARISON OF LOOPS

• **for loop** – used when you know how many times to execute or each iteration has a natural increment, e.g., iterating over a sequence

• **while loop** – Pre-condition check, i.e., used to execute 0 or more times

• **do-while loop** – Post-condition check, i.e., used to execute 1 or more times
• **break** — immediately exit the loop. Do not continue executing any more of the loop:

```java
while (true) {
    if (q-key-is-pressed())
        // quit the game
    break;
    Game-loop();
}
```

• **continue** — immediately skip to the end of the body of the loop, i.e., start next iteration (checking the condition):

```java
for (int i = 0; i < 10; ++i) {
    // OCD against prime numbers
    if (isPrime(i))
        continue;
    HandleNotPrimes();
}
```
FUNCTIONS AND METHODS

FUNCTIONS AND METHODS IN PSEUDOCODE, PYTHON, AND JAVA
METHODS

• A **method** is a collection of statements that are grouped together to perform an operation.
  • Often called a “function,” “subroutine,” or “algorithm”

• You can think of the method body as a black box that contains the detailed implementation for the method

• Methods can be used to reduce redundant coding and enable code reuse

• Methods can also be used to modularize code and improve the quality of the program.

\[ f(x, y, z) \]
METHODS

- **Method signature** is the combination of the method name and the parameter list
  - Method **overloading** is creating more than one method with the same name but different signatures
- The variables defined in the method header are known as **formal parameters**
- When a method is invoked, you pass a value to the parameter. This value is referred to as **actual parameter** or **argument**

- A method may or may not return a value
- What happens when a function is called:
  - Control transfers to the function code.
  - Argument variables are assigned the values given in the call.
  - Function code is executed.
  - Return value is assigned in place of the function name in calling code.
  - Control transfers back to the calling code.
- Leads to notion of a **call stack**, and allows for another way to alter flow of execution
- For pseudocode, Python*, and Java, parameters are passed **by-value**
METHODS

PSEUDOCODE

Algorithm FunctionName
Input: FormalParameterList
Output: ReturnType explanation (optional)
1. {Do some operations}
2. {Use return when there is output}

PYTHON

1. def FunctionName(FormalParameterList):
2. # Do some operations
3. # Use return when there is output

JAVA CODE

1. public static ReturnType FunctionName(FormalParameterList) {
2.   // Do some operations
3.   // Use return when ReturnType is not void
4. }

- FormalParameterList is comma separated and declares each variable
- Methods must be inside of a class in Java only
- void denotes when there is no output in Java only
CREATING AND USING OBJECTS

IN PSEUDOCODE, PYTHON, AND JAVA
OBJECTS

• An **object** represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects. An object has a unique identity, state, and behaviors.
  
  • The **state** of an object consists of a set of **data fields** (also known as **properties**) with their current values.
  
  • The **behavior** of an object is defined by a set of methods.
  
  • Essentially, it is a user defined type
    
    • An object **class** defines its possible states and its behaviors
    
    • An object **instance** is a variable of the object type, i.e., it is a specific “value” or state
## OBJECT

### CREATION AND USE

<table>
<thead>
<tr>
<th>PSEUDOCODE</th>
<th>PYTHON</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. <strong>DataType</strong> x initialized with <em>values</em></td>
<td>1. x = <strong>DataType</strong>(values)</td>
</tr>
<tr>
<td>2. x.<em>methodName</em>(<em>actualParameters</em>)</td>
<td>2. X.<em>methodName</em>(actualParameters)</td>
</tr>
<tr>
<td>Example</td>
<td>Example</td>
</tr>
<tr>
<td>1. <strong>Circle</strong> c with radius of 10</td>
<td>1. c = <strong>Circle</strong>(10)</td>
</tr>
<tr>
<td>2. <strong>Real</strong> a ← c.<em>area</em>()</td>
<td>2. a = c.<em>area</em>();</td>
</tr>
</tbody>
</table>

### JAVA CODE

| 1. **DataType** x = **new** **DataType**(values);                         | 2. x.*methodName*(*actualParameters*);                                 |
| Example                                                                   |                                   |
| 1. **Circle** c = **new** **Circle**(10);                                 | 2. **double** a = c.*area*();                                           |
EXERCISE

• With your partner, write algorithms in pseudocode, in java, and as a bonus in python to:
  • Output the first \( n \) numbers in the series: \( \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6}, \ldots \)
  • Find the \( n \)th prime number
  • Determine if two circles overlap
    • Assume a reasonable Circle class already exists
Summary

• Computer science is the study of algorithms
  • Pseudocode is an effective tool to communicate algorithms
  • Programming, e.g., with Java, is a means of implementing algorithms

• Java Primer
  • Program structure and variables
  • Control flow
  • Functions and Methods
  • Using libraries and objects