CSCI 200: Foundational Programming Concepts & Design Lecture 13

Object-Oriented Programming:

Managing & Encapsulating State

9/22 In Class Survey

Access Code: meteor

Learning Outcomes For Today

- Discuss the concept of encapsulation
- Draw a class diagram using UML to describe the structure of a class and its members
- Discuss the difference between a class and an object
- Create a class containing data members and member functions
- Compare and contrast Procedural Programming with Object-Oriented Programming

Learning Outcomes For Today

- Explain the following terms and how they are used
  - (1) dot operator / member access operator
  - (2) data member
  - (3) scope resolution operator
- Discuss the difference between
  - (1) a class and an object
  - (2) a class and a struct
- Discuss the concept of scope within and outside a class & struct

On Tap For Today

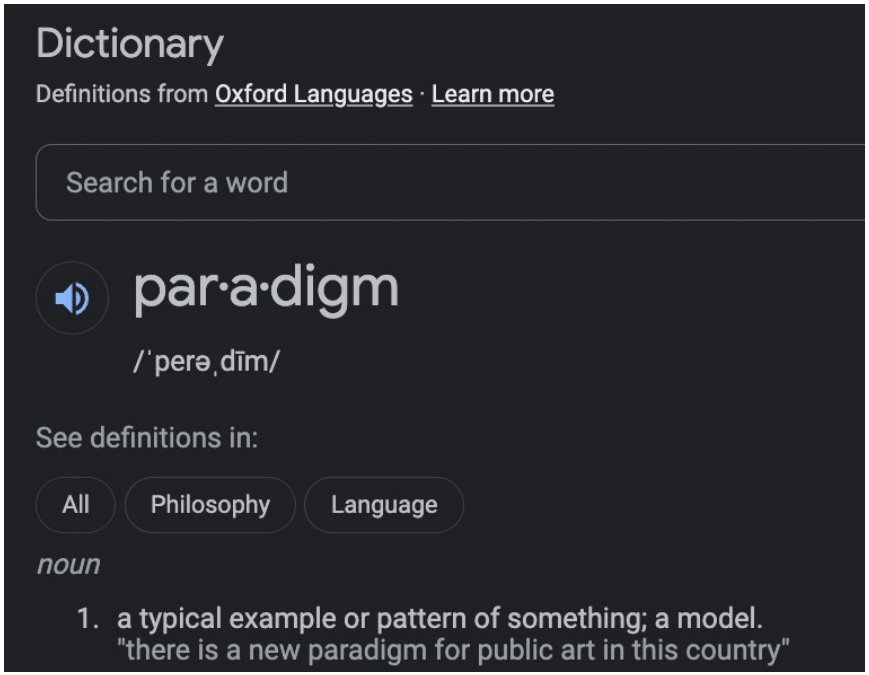
- Programming Paradigms
  - Imperative Programming
- Object-Oriented Programming
  - Classes & Objects
- Practice

On Tap For Today

- Programming Paradigms
  - Imperative Programming
- Object-Oriented Programming
  - Classes & Objects
- Practice

Programming Paradigm

- According to Google:
  - *Paradigm*: a typical example or pattern of something



Programming Paradigm

- According to Google:
  - *Paradigm*: a typical example or pattern of something
- Programming Paradigm: a style, or way, of programming
  - Independent of a programming language
  - A language can exhibit many paradigms

On Tap For Today

- Programming Paradigms
  - Imperative Programming
- Object-Oriented Programming
  - Classes & Objects
- Practice

Imperative Programming

- Root word: *impero*
  - Latin for “I command”

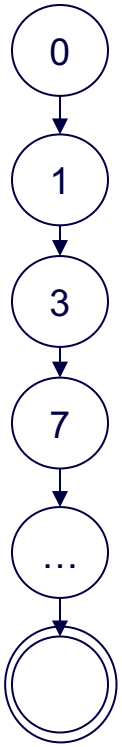
- You are the emperor giving orders to the computer

Imperative Programming

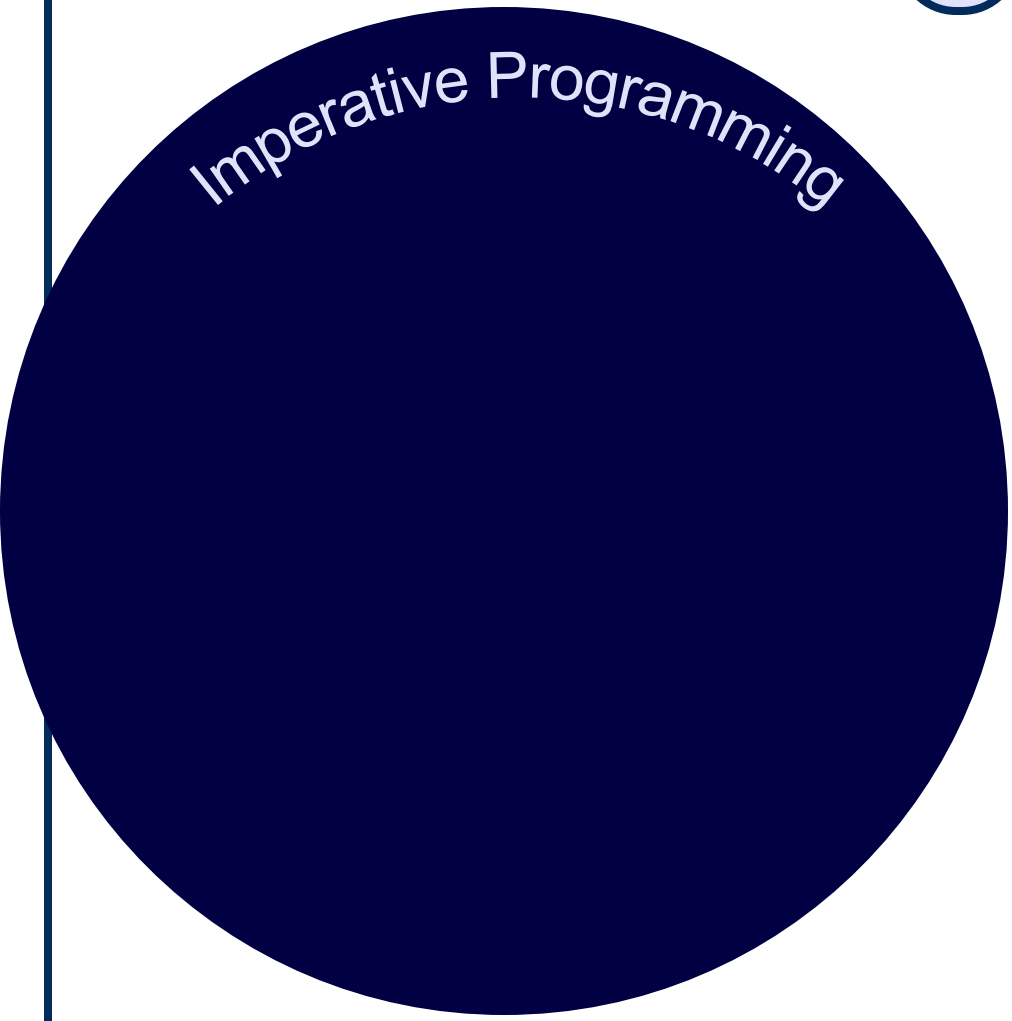
- Explicit sequence of steps to perform one at a time
  - Shows how the computation takes place
- Each step changes the **state** of the program
  - **state** comprised of stack information
    - Current line of execution
    - Variables that are in scope

Imperative Programming

```
int main() {
  int sum = 0;
  sum += 1;
  sum += 2;
  sum += 3;
  sum += 4;
  sum += 5;
  sum += 6;
  sum += 7;
  sum += 8;
  sum += 9;
  sum += 10;
  cout << "The sum is: " << sum << endl;
  return 0;
}
```

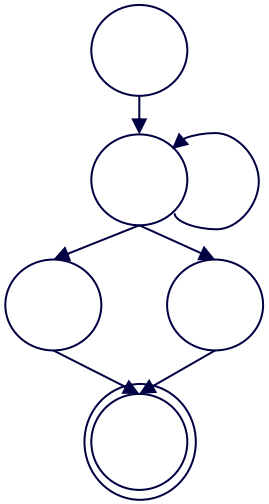


Programming Paradigms



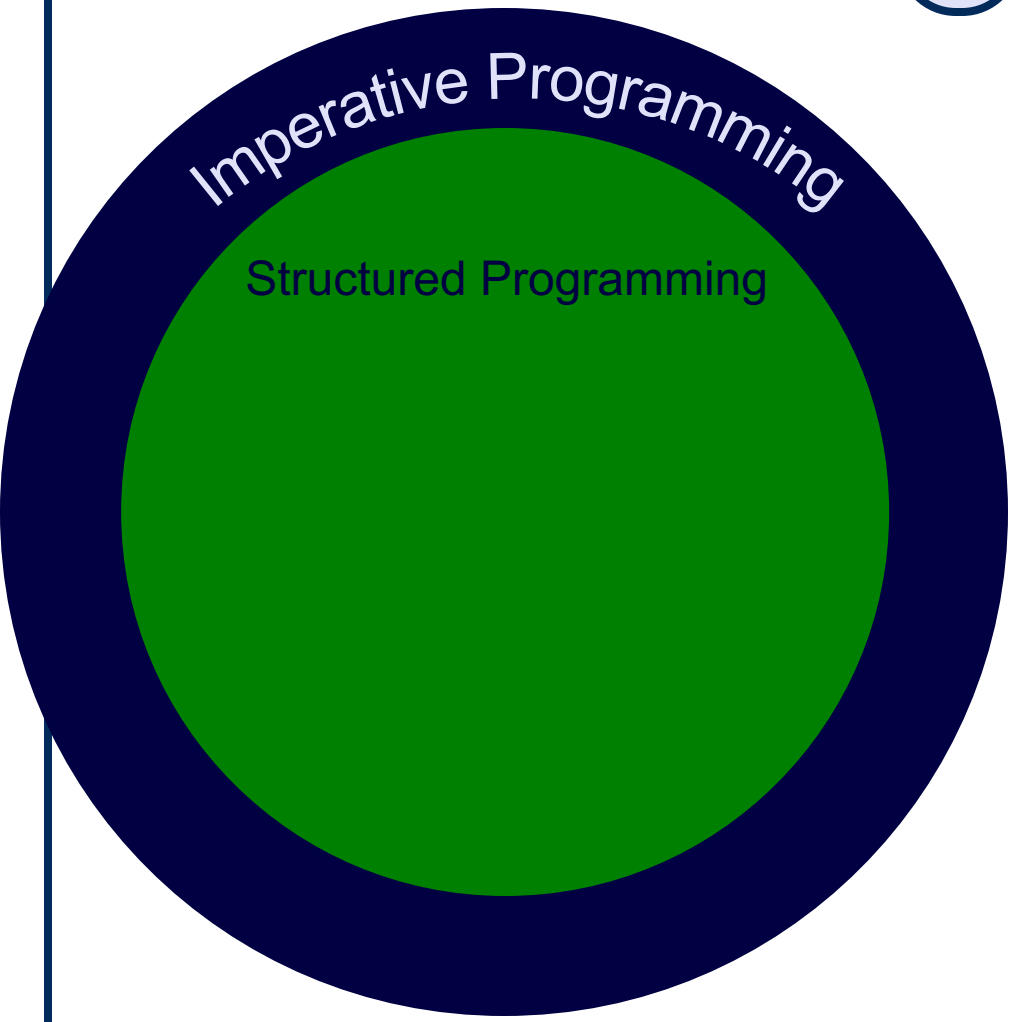
Structured Programming

- Imperative Programming where flow is defined by control structures (loops, conditionals)



```
int main() {
  int sum = 0;
  for(int i = 1; i < 10; i++) {
    sum += i;
  }
  cout << "The sum is: " << sum << endl;
  return 0;
}
```

Programming Paradigms



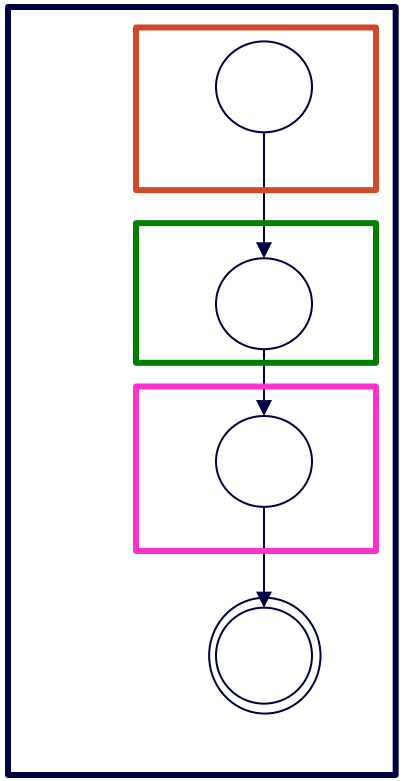
Procedural Programming

- Imperative Programming where program state is manipulated by sequence of subroutine procedure calls

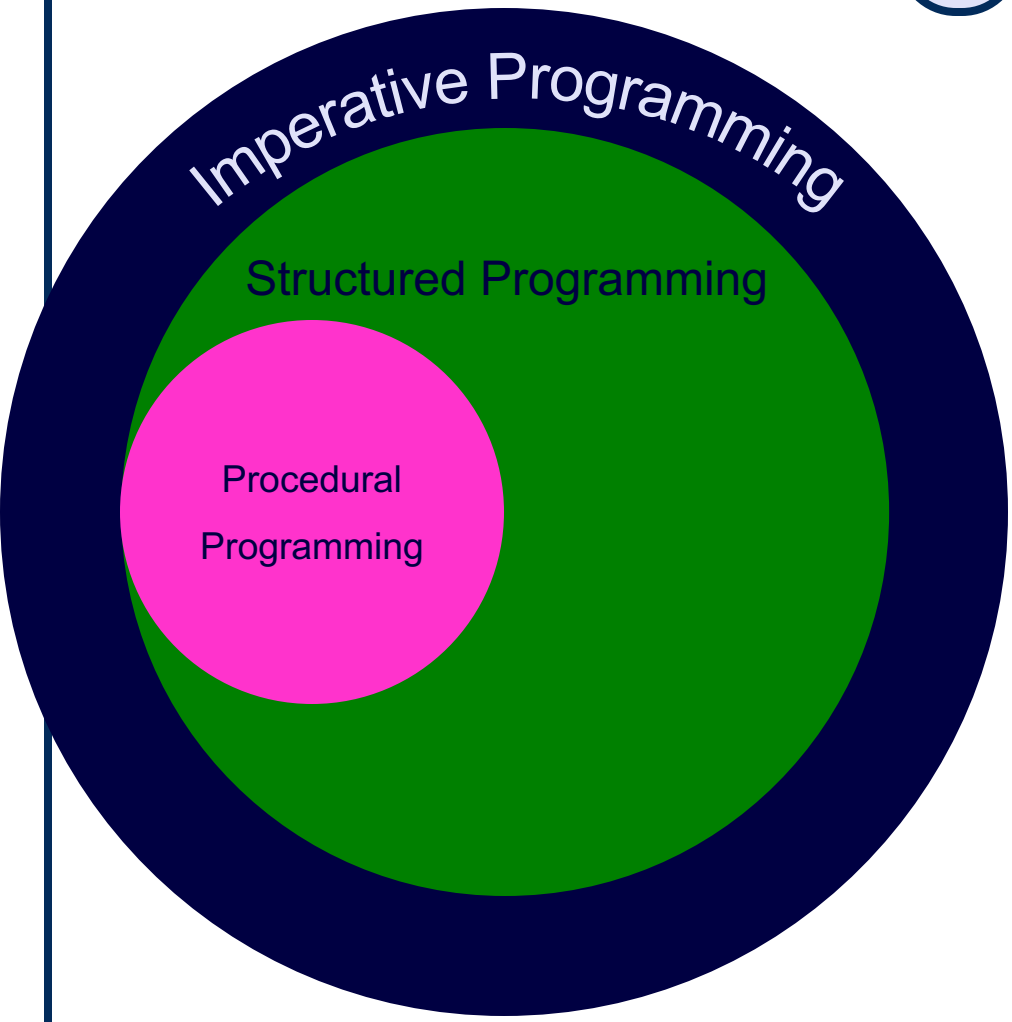
- **Note**: procedures are implemented as functions, but do not return a value. Rather, perform a task and generate a desired side effect

Procedural Programming

```
void init_int(int* const P_arg, const int VAL) { *P_arg = VAL; }

void add(const int A, const int B, int* const P_sum) { *P_sum = A + B; }

void int_summation(const int M, const int N, int* const P_result) {
  for(int i = M; i <= N; i++) {
    add(*P_result, i, P_result);
  }
}

void print_int_sum(const char* MSG, const int SUM) {
  cout << MSG << SUM << endl;
}

int main() {
  int sum;
  init_int( &sum, 0 );
  int_summation( 1, 10, &sum );
  print_int_sum( "The sum is: ", sum );
  return 0;
}
```



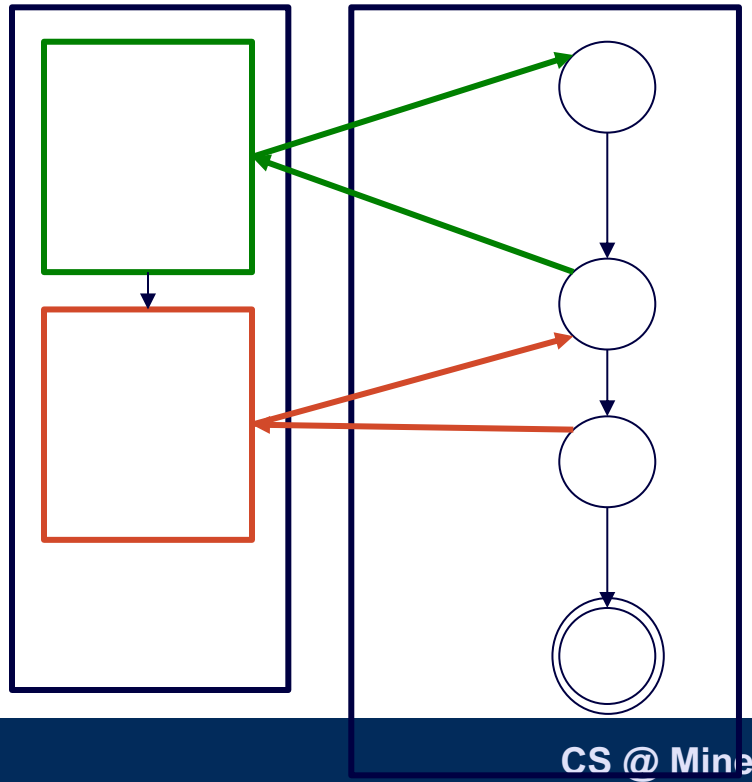
Programming Paradigms



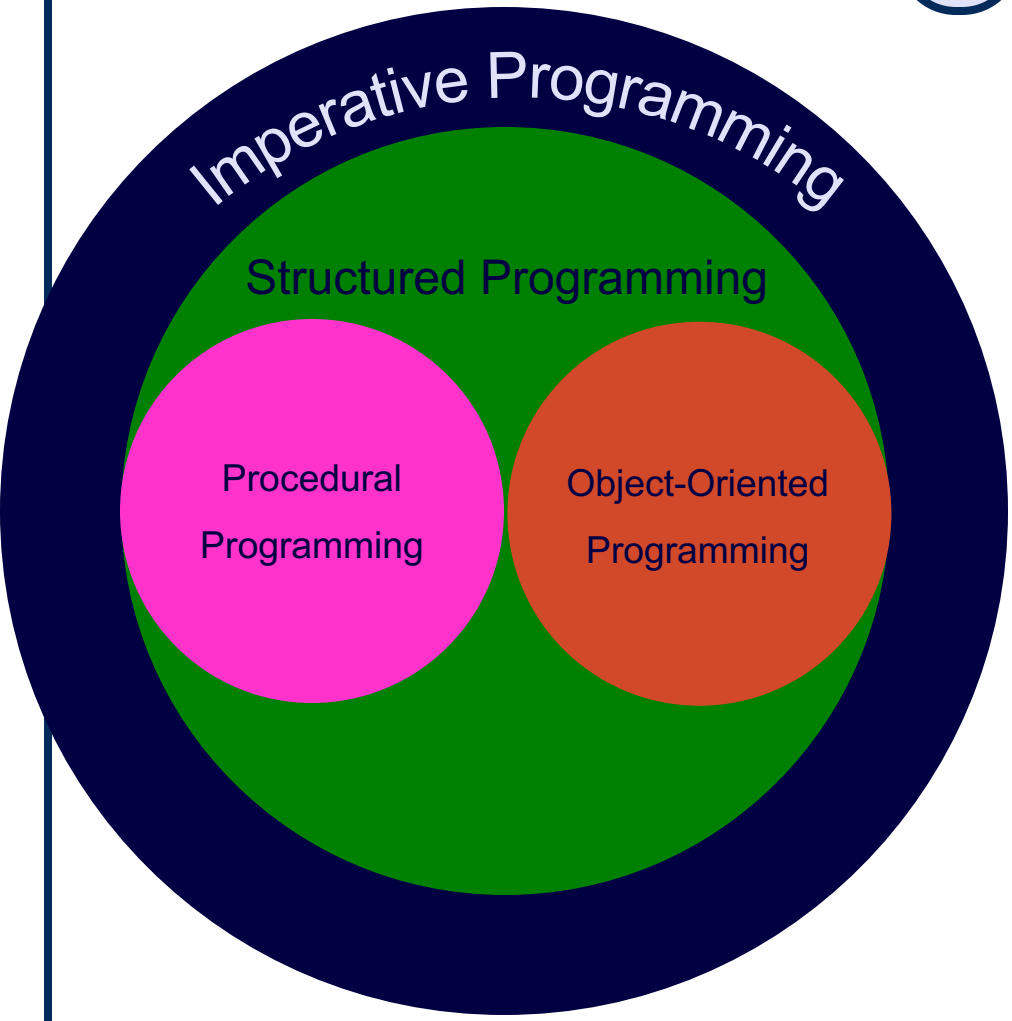
Object-Oriented Programming

- Imperative Programming where program state is encapsulated in a series of objects
  - Only objects can manipulate their own state

```
int main() {
  SumMachine summer;
  summer.reset( 0 );
  summer.setRange( 1, 10 );
  summer.sum();
  cout << "The sum is: "
       << summer.getSum() << endl;
  return 0;
}
```



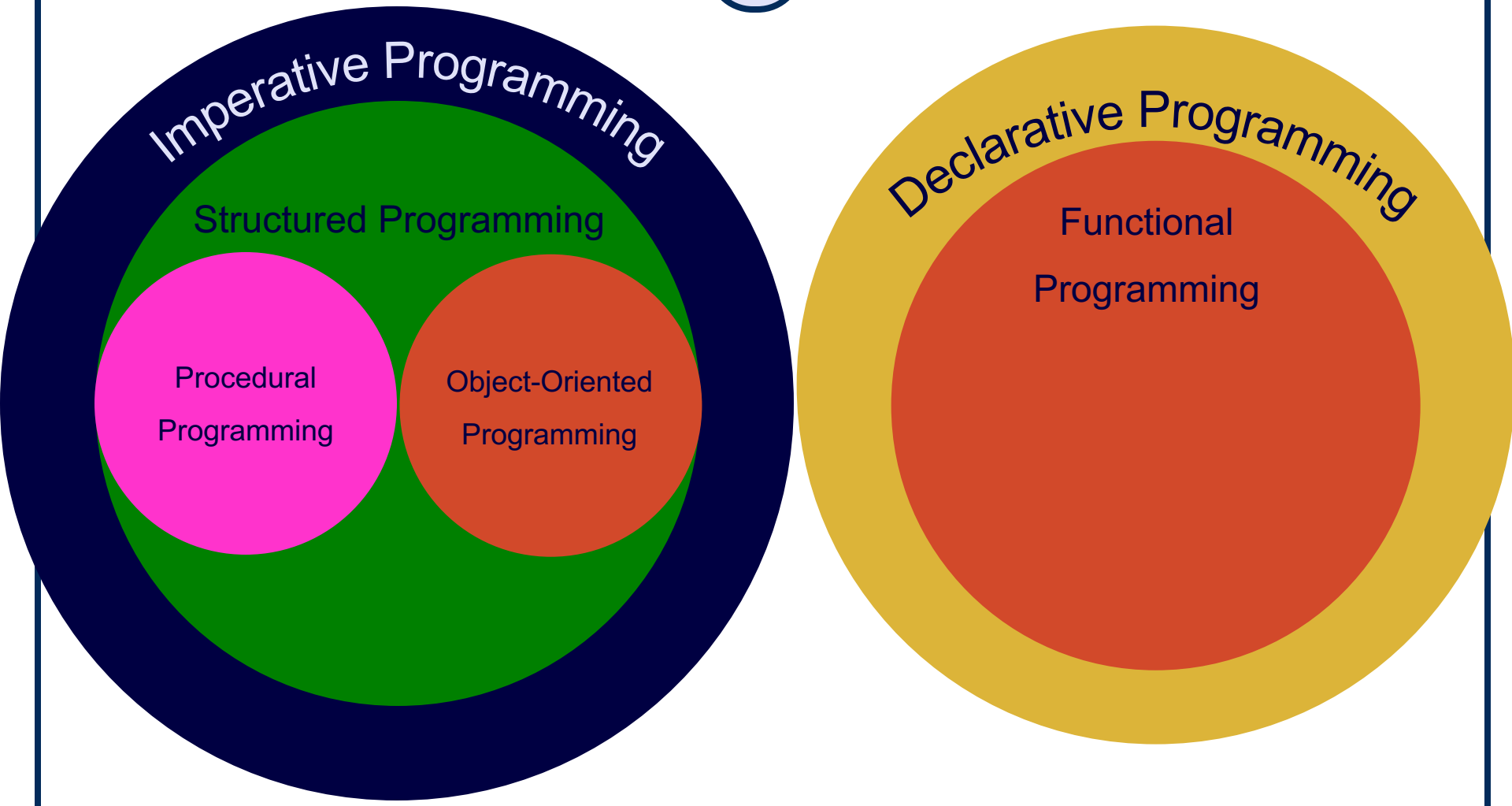
Programming Paradigms



Programming Paradigms

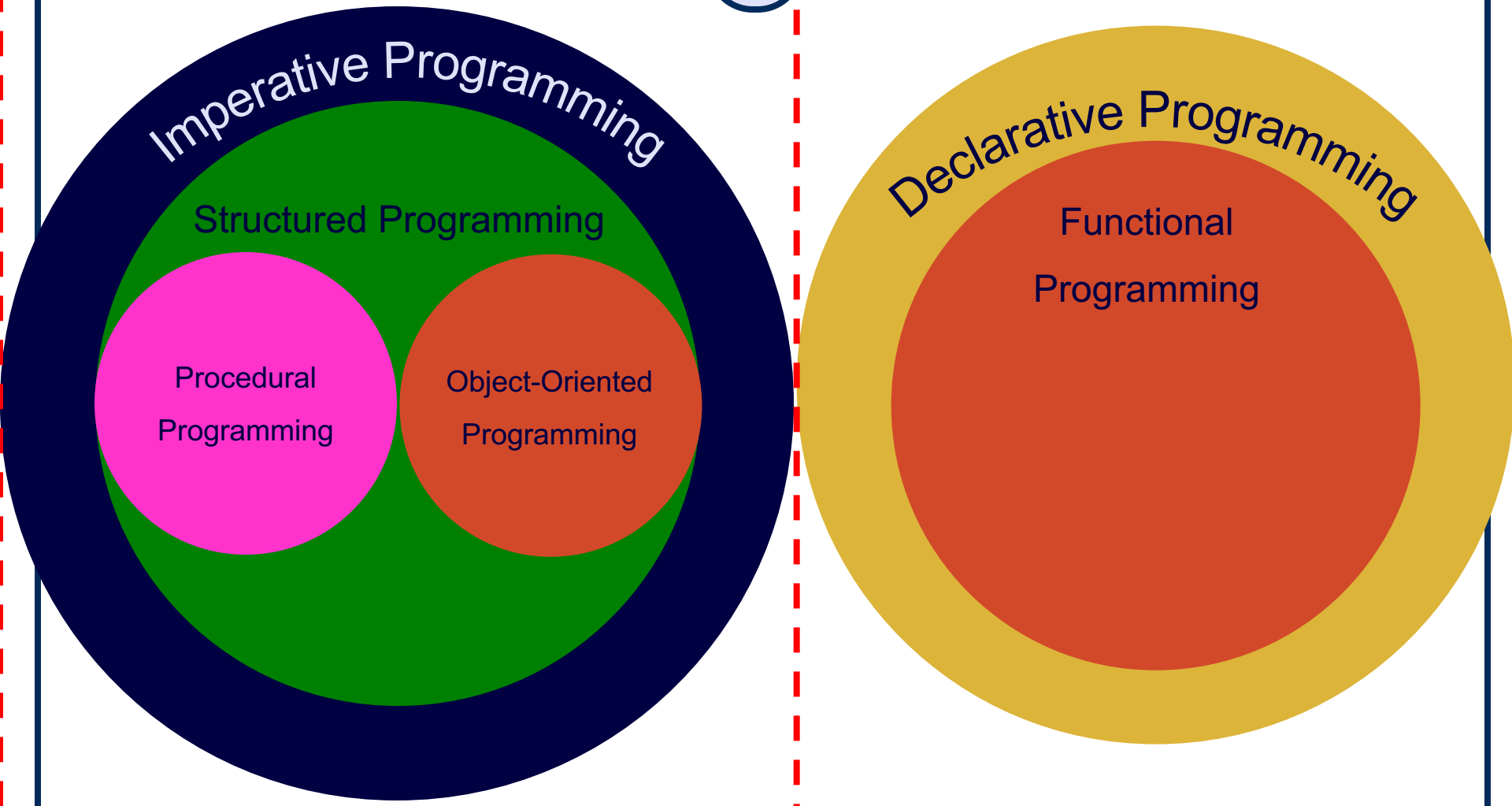
- Imperative Programming
  (how to do something)
  - Structured Programming
    - Procedural Programming
    - Object-Oriented Programming
- Declarative Programming
  (what result looks like, but not how to compute it)
  - Functional Programming
- And others

Programming Paradigms

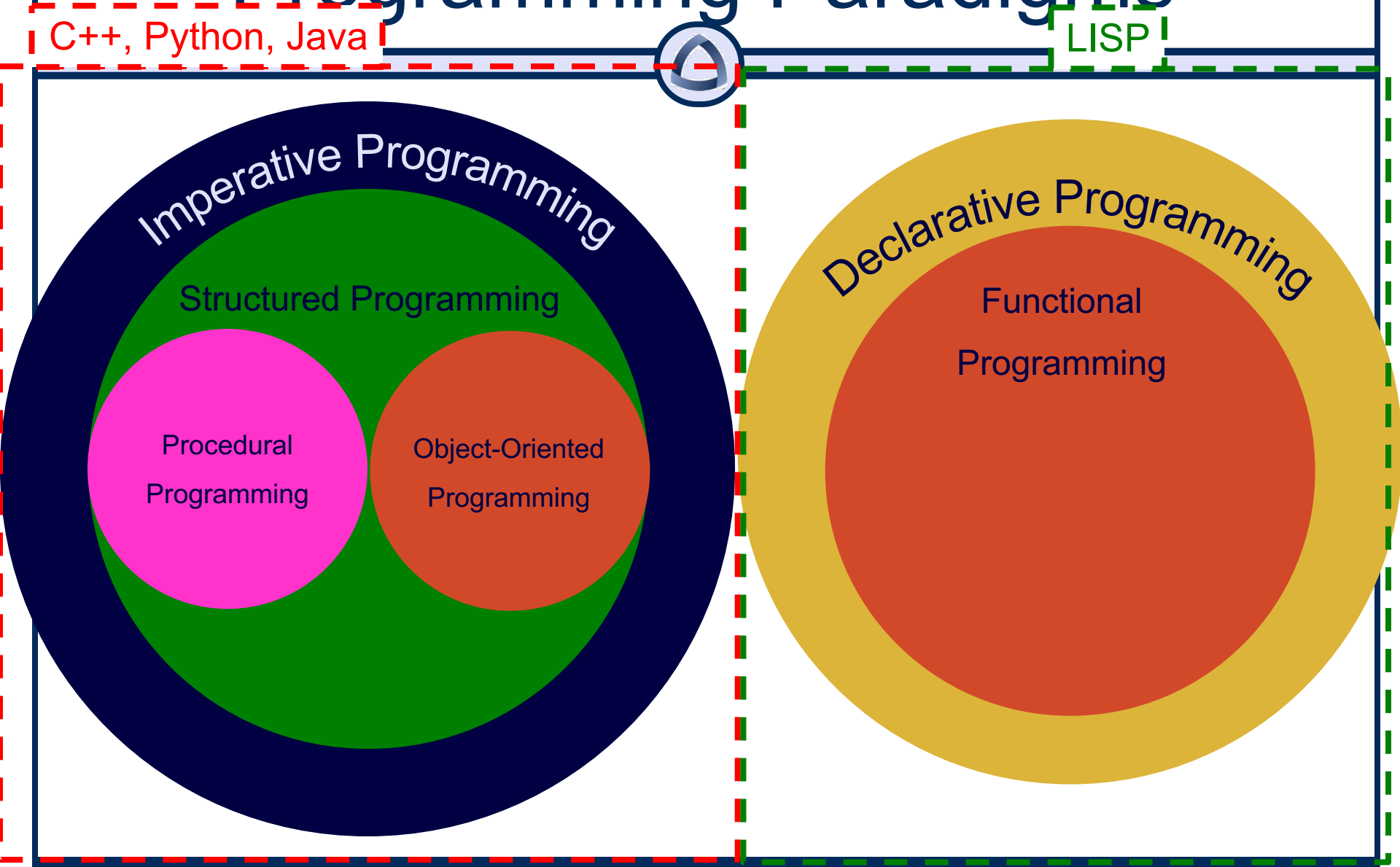


Programming Paradigms

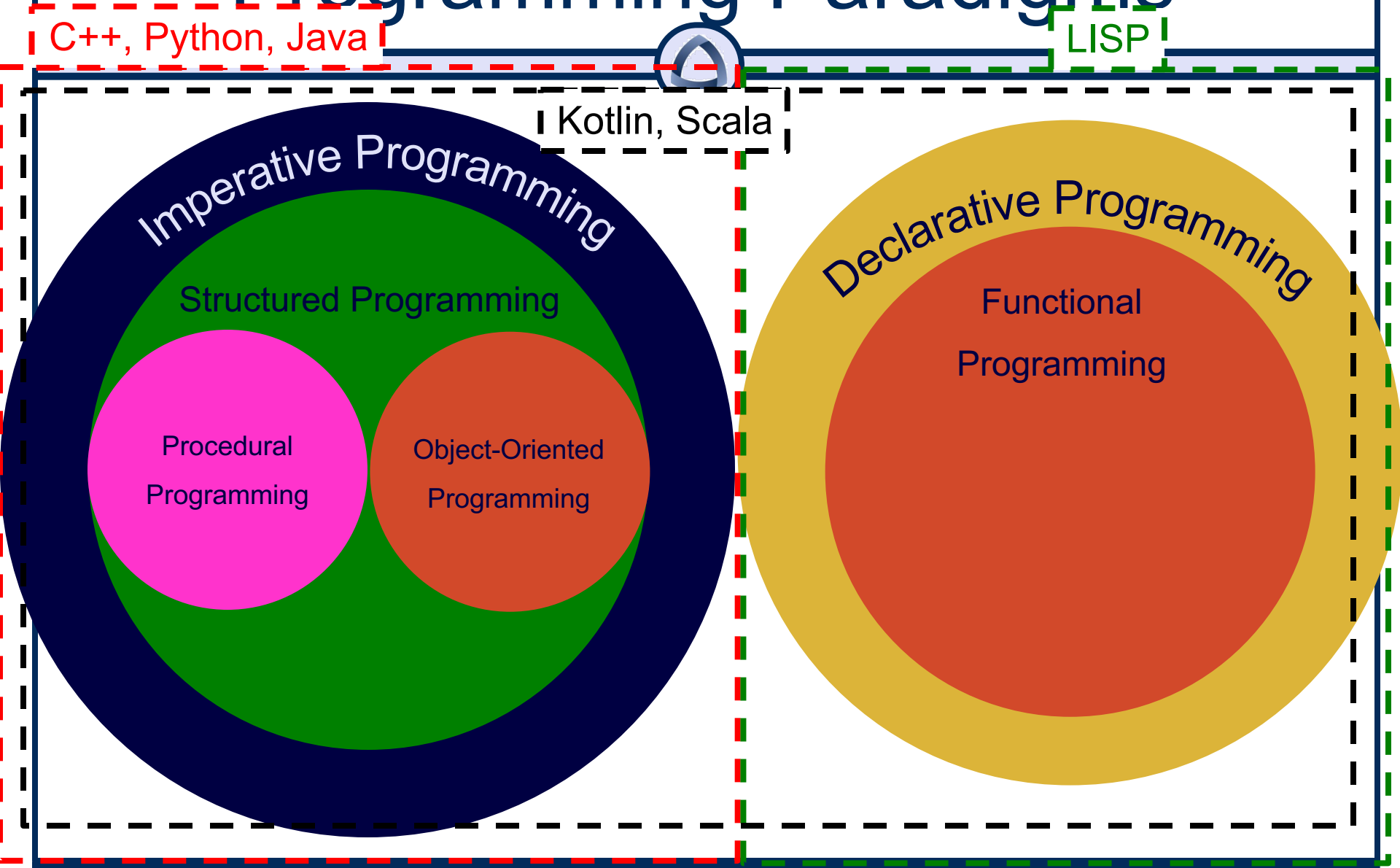
C++, Python, Java



Programming Paradigms



Programming Paradigms



On Tap For Today

- Programming Paradigms
  - Imperative Programming
- Object-Oriented Programming
  - Classes & Objects
- Practice

How do we tell the computer about things?

- Variables (int, double, char, etc.)
- *Data structures (array/vector, struct)*
- What else?
  - ANYTHING ELSE!
    - Flower
    - Dog
    - Car
    - Rock
    - Computer

Practice: Tyrannosaurus Rex

- Describe a T-Rex
  - What are its attributes?
  - What can it do?

Object-Oriented Programming (OOP)

- Programming paradigm that groups like attributes and behaviors into a class

- Program to the domain
  - Use terminology and objects that are present in the field

Object-Oriented Programming (OOP)

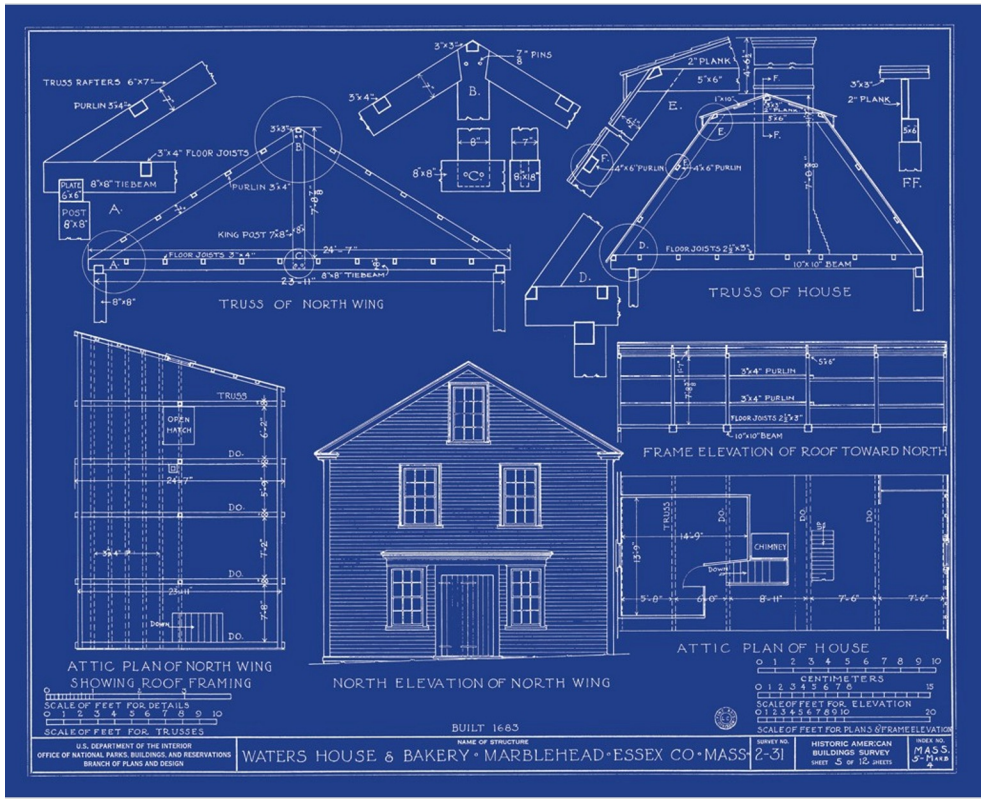
- Programming paradigm that groups like attributes and behaviors into a class
- Class: defines a new data type or adds functionality to an existing data type
- Object: variable/instance of a defined class

Representing other things

- Create a **class** to represent a complex thing
  - A class **encapsulates** attributes (variables) and behaviors (functions) of real world things
  - Attributes
  - Behaviors
  - Abstraction!

Classes & Objects

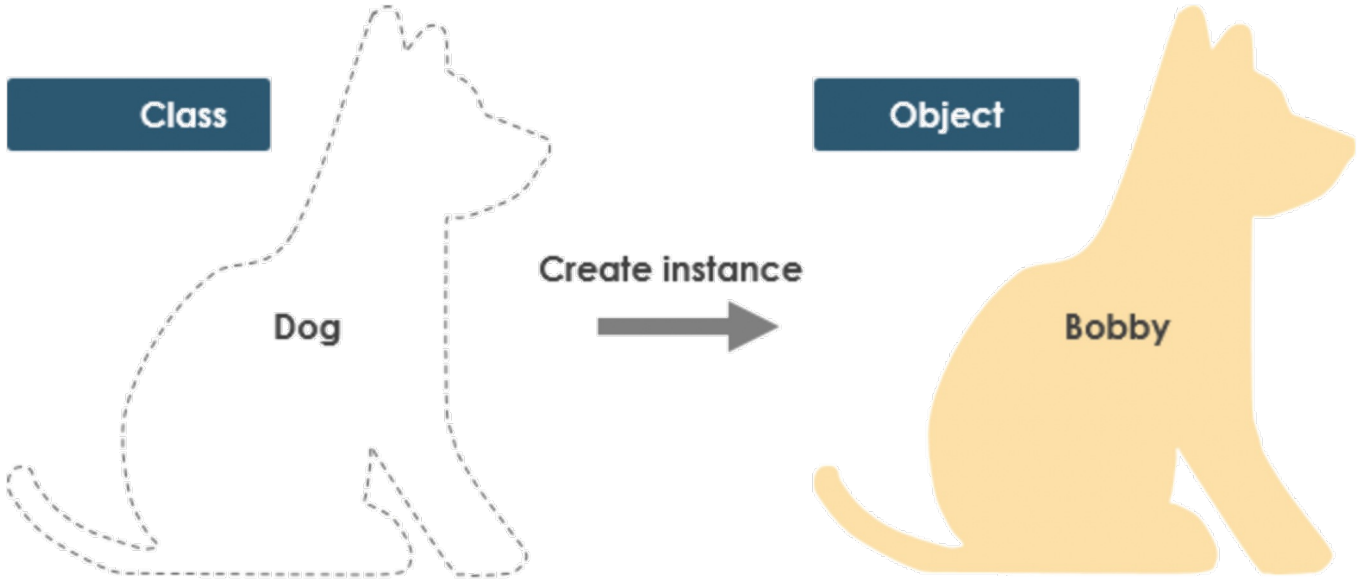
- A `class` is a blueprint



Classes & Objects

- An `object` is an instance of the `class`
- An `object` provides values for the data members of a `class`

Class & Object Example



Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

Property Values	Methods
Color: Yellow	Sit
Eye Color: Brown	Lay Down
Height: 17 in	Shake
Length: 35 in	Come
Weight: 24 pounds	

Creating a Class Diagram

- Uses Unified Modeling Language (UML) to show structure of a class
- List attributes and behaviors of a class

ClassName
attrName1 : attrType1 attrName2 : attrType2 attrname3 : attrType3
behavior1() : returnType1 behavior2() : returnType2 behavior3(params) : returnType3

TyrannosaurusRex
species : string height : double weight : double
run() : void eat(Meat) : void roar() : string

Practice: Pterodactyl

- Describe a Pterodactyl
  - What are its attributes?
  - What can it do?

Object-Oriented Programming

- Classes exhibit
  - "Has-A" relationships with its own attributes & state
    - Our focus for now
  - "Is-A" relationships with common ancestors that share attributes & state
    - Coming after Exam II (btw that's Oct 25)

On Tap For Today

- Programming Paradigms
  - Imperative Programming
- Object-Oriented Programming
  - Classes & Objects
- Practice

Classes & Objects

- Class declares the **data members** & **methods** of an object
  - Data members represent the current state of the object

- Class made up of declaration + definition

Class Declaration Syntax

```
class ClassName {
public:                 // we'll cover this later on
    dataType variableName;
    dataType variableName2;
    returnType functionName( params );
};
```

Class Declaration

- Place in its own header file ClassName.h

```
// inside Box.h
#ifndef BOX_H
#define BOX_H
class Box {
public:
    float height;
    float depth;
    float width;

};
#endif
```

Creating an Object

```
// inside main.cpp
#include "Box.h"

int main() {
    Box smallBox;
    smallBox.height = 4;
    smallBox.width = 4;
    cout << "Enter the box length: ";
    cin >> smallBox.depth;
    cout << "The volume is: ";
    cout << smallBox.width * smallBox.height * smallBox.depth << endl;
    return 0;
}
```

Class Declaration

- Place in its own header file ClassName.h

```
// inside Box.h
class Box {
public:
    float height;
    float depth;
    float width;
    float volume();
};
```

- Will get to implementation next

Class Definition

- Placed in a class implementation file ClassName.cpp

```
// inside Box.cpp
#include "Box.h"


float Box::volume() {
    return height * depth * width;
}
```

- Functions have access to ALL data members of a class

- :: is the **Scope Resolution operator**
  - Specifies which scope an identifier belongs to
- In our case, which class a function belongs to
  - Box::volume()
- Could have two classes with the same function name
  - Box::volume() Tube::volume()

Precedence	Operator	Associativity
1	Parenthesis: (a)	Innermost First
2	Scope Resolution: S::	Left to Right
3	Postfix Unary Operators: a++ a-- f() a.	
4	Prefix Unary Operators: ++a --a +a -a !a (type)a &a *p new delete	Right to Left
5	Binary Operators: a*b a/b a%b	Left to Right
6	Binary Operators: a+b a-b	
7	Relational Operators: a<b a>b a<=b a>=b	
8	Relational Operators: a==b a!=b	
9	Logical Operators: a&&b	
10	Logical Operators: a||b	
11	Assignment Operators: a=b a+=b a-=b a*=b a/=b a%=b	Right to Left

Creating an Object

```
// inside main.cpp
#include "Box.h"

int main() {
    Box smallBox;
    smallBox.height = 4;
    smallBox.width = 4;
    cout << "Enter the box length: ";
    cin >> smallBox.depth;
    cout << "The volume is: ";
    cout << smallBox.volume() << endl;
    return 0;
}
```

On Tap For Today

- Programming Paradigms
  - Imperative Programming
- Object-Oriented Programming
  - Classes & Objects
- Practice

To Do For Next Time

- Be completing Set2
- Keep going with zyBooks