|  |  |
| --- | --- |
|  | |
| * What is object-oriented programming? * What languages support OOP? * Angular * What are the four core concepts of object-oriented programming?  * What is procedural Programming?     * Spaghetti Code * OOP vs Procedural   * What are Properties? * What are Methods? * What is Encapsulation?   * What is Abstraction?    * Inheritance       * Polymorphism          * Benefits of OOP         * What are objects?  * How do you create an object? * What is an object literal?              * What are factories?    * What is behavior? * What is a constructor?   * What is a constructor property? * Functions are Objects * What is a primitive type? * What are Reference types?  * Adding/Removing Properties * How to work with the properties of an object? * How do you define private property? * What is a getter? * What is a setter? | * A programming style centered around objects rather than functions. * C#, Java, Ruby, Python, JavaScript and more * An object-oriented programming framework * 1. Encapsulation * 2. Abstraction * 3. Inheritance * 4. Polymorphism * Predates OOP as a legacy style of programming. * Divided a program into a set of functions with data stored in variables.   + Simple and straightforward.   + Common practice in beginning programming subjects. * Code that is interdependent and can break in multiple areas if a modification is made. * Procedural divides program by individual function parts * Procedural programming separates (decouples) variables from functions. * OOP combines related variables and functions into a unit called an ‘object’. * Groups related variables and the functions that operate on them into objects.  |  |  | | --- | --- | | * Procedural   let base\_Salary = 30\_000;  let OT = 10;  let rate = 20;  function get\_Wage(baseSalary, OT, rate) {  return base\_Salary + (OT \* rate);  } | * OOP   let employee = {  base\_Salary = 30\_000,  OT = 10,  Rate = 20,    get\_Wage: function() {  Return this.base\_Salary + (this.OT \* this.rate);  } |  * Object variables. * Object functions. * The grouping of variables(properties) and functions(methods) into a singular object. * The concept of OOP is to encapsulate functions to limit the number of parameters. * Hiding complex functionality behind simple interfaces. * Hiding properties and methods from outside influence.   + Makes interfaces simpler.   + Helps reduce the impact of change (spaghetti code). * A mechanism that allows you to eliminate redundant code. * Allows for a template object to be created and all related objects inherit methods and properties from the template object to reduce redundant code.   + [Ex] HTML\_Element can be a generic object for textbox, select(drop down list), & checkboxes * Translates to ‘many forms’. * Allows you to get rid of long if / else and switch / case statements.  |  |  | | --- | --- | | * Procedural   Switch(…) {  case ‘select’ : renderSelect();  case ‘text’: renderTextBox();  case ‘checkbox’: renderCheckbox(); | * OOP   element.render();//gets added to select, textbox, and checkbox |  * ***Encapsulation*** groups related variables and functions together reducing complexity & increasing reusability * ***Abstraction*** hides details and complexity & shows only the essentials reducing complexity and isolates impact of change throughout the program. * ***Inheritance*** eliminates redundant code by promoting the use of a generic template object and having other objects inherit from it * ***Polymorphism*** allows switch / case and if / else statements to be refactored to a single statement.     * A syntax used for defining key value types, properties, and methods of an object. * Const circle = {   radius: 1,  location: {  X: 1,  Y: 2  },  draw: function() {  console.log  }  }   * + The circle object has three members:     - Radius, location & draw.   + Function members are methods.     - Methods define logic.   + Variable members are properties.     - Properties hold key value pairs. * A reusable function syntax for objects with behavior   function createCircle(radius) {      return {          radius: radius,          draw: function() {              console.log('draw!');          }      };  }  const circle2 = createCircle(5);  console.log(circle2    * When an object has more than one method * A reusable function syntax for objects   function Circle(radius) {      console.log('this', this)      this.radius = radius;      this.draw = function() {          console.log('draw!!')      }  }  const circle3 = new Circle(10);  console.log(circle3);    * References the function that was used to create the object (new String();, new Boolean();, new Object();))          * Value types * Number, string, Boolean, symbol, undefined, null * Copied by their value. * Object, function, and arrays * Copied for their reference. |
| Summary | |