1 **Objects**

JavaScript

2 **Creating Objects**

- The new operator instantiates a new object
- Keyword "new" must be followed by a special function called a constructor which initializes the object
- Format:

  var *object* = new *Constructor*([*parameter1*, …]);
- Examples:

  var d = new Date("1 Jan 2017");

  var a = new Array(10, 15, 20, 25, 30, 35, 40, 45, 50); // *parameters* are a list of array element values

3 **Try It Out**

- objects1.htm

4 **Programmer-Defined Functions (Page 1)**

- A function is a block of code that performs a particular task (like methods in Java)
- The function is executed when it is "invoked" (called)
- Optional parameters in the header are declared with any type, not even var
- Once the function has concluded executing, it may have a result (return value)

5 **Programmer-Defined Functions (Page 2)**

- Format:

  function *functionName*([*parameter1*, *parameter2*, …])

  { … }
- Example:

  function createCircle(x, y, radius)
  - Programmer-defined functions often are placed inside a <script> block in the <head> element and called from a <script> block in the <body> element

6 **Try It Out**

- objects2.htm

7 **Objects (Page 1)**

- A programmer-defined object is a structure from which new objects can be instantiated
  - Like a class in Java
  - Multiple objects can be instantiated from the same programmer-defined object
- An object literal within the object is a comma-delimited set of colon-separated *name*:*value* pairs in the {braces} of body of the object
  - Each *name* in the object literal is a property of the object

8 **Objects (Page 2)**

- Format:

```
var object =
{
    name1: value1,
    name2: value2, …
};
```

  - Each *name* is a "property" (a programmer-defined identifier like a variable or function name)
  - *value* is the value assigned to the *name* (could be either a *parameter* or a *literal*)

9 **Objects (Page 4)**

- Example:

```
var person =
{
    firstName: "Harry",
    lastName: "Evans"
};
```

10 **Properties**

- The *names* in an object are its properties
- They are accessible using dot (.) notation (like the length property of a string) e.g.: *object.property*
- Examples:

```
person1.firstName
aCircle.radius
```

11 **Functions with Objects (Page 1)**

- In JavaScript objects are placed *inside functions*
- The called function *returns* the object at conclusion of its execution

12 **Functions with Objects (Page 2)**

- Format:

```
function functionName( [parameters] )
{
    var objectVariable =
    {
        name1: value1,
        name2: value2, …
    };
    return objectVariable;
}
```

13 **Functions with Objects (Page 3)**

- Example 1:

```
function createCircle(x, y, radius)
{
    var circle =
    {
        x: x,
        y: y,
        radius: radius
    };
    return circle;
}
```

14 **Functions with Objects (Page 4)**

- Example 2:

```
function createPerson(firstName, lastName)
{
    var person =
    {
        firstName: firstName,
        lastName: lastName
    };
    return circle;
}
```

15 **Instantiating Objects**

- To instantiate a programmer-defined object, call the programmer-defined function that contains the object:
  var *variable* = *functionName*([*parameters*]);
- Examples:
  var aCircle = createCircle(10, -15, 25);
  var person1 = createPerson("Harry", "Evans");

16 **Try It Out**

- objects3.htm

17 **Functions in Object Literals (Page 1)**

- A *function* may be defined in the object literal
- Format:

```
var object =
{
    functionName: function( [parameter1, ... ] )
```

```
    {
       statements…
    }
};
```

▫ Parameters may be defined in the function header

18 **Functions in Object Literals (Page 2)**

• Example:

```
var person =
{
   firstName: "Harry",
   lastName: "Evans",
   getFullName: function()
   {
       return firstName + "  "  + lastName;
   }
};
```

▫ The function is *comma-delimited* from the other elements of the object literal

19 **Functions in Object Literals (Page 3)**

• Format for calling the object function:
objectName.functionName([*parameters*])

• Example to instantiate the object and call one of its functions:

```
var person = createPerson();  // Instantiate the object
document.write( person.getFullName() );
```

20 **Try It Out**

• objects4a.htm
• objects4b.htm

21 **Properties (Page 1)**

• A property is updated (*set*) when it appears on the left side of the assignment (=) operator:
object.property = newValue;

• Example:

```
aCircle.x = 20;
```

•

22 **Properties (Page 2)**

• A property is accessed (*get*) when it appears on the right side of the assignment (=) operator:
variable = object.property;

• Examples:

```
var a = aCircle.x;
```

23 **Properties (Page 3)**

- A property also is *retrieved* (accessed) (*get*) when it appears as part of a larger expression
- Examples:

```
document.write(aCircle.x);
if (aCircle.radius >= 20) ...
```

24 **Try It Out**

- objects5.htm

25 **Properties (Page 4)**

- Properties also may be accessed (updated and retrieved) using *array notation*, e.g. [brackets] around property name as a string
- Format:

object["propertyName"]

- Examples:

```
aCircle["x"] = 20;                // set
var x = aCircle["x"];             // get
document.write( aCircle["x"] ); // get
```

26 **Try It Out**

- objects5a.htm

27 **The create() Method for Objects (Page 1)**

- The createObject() method is an alternative option to the new operator for creating new objects
- It instantiates an object from a prototype
- The prototype is a previously defined object with properties and functions

28 **The create() Method for Objects (Page 2)**

- Format:

var objectVariable = Object.create(*prototype*);

- Example:

```
Car =
{
   description: "",
   year: "",
   color: ""
};
...
var car = Object.create(Car);
```

29 **Try It Out**

- objects6.htm
- objects6a.htm

30 **The delete Operator (Page 1)**

- The delete operator for the property of an object deletes the property from the object
- Accessing the property afterwards returns a result of undefined

31 **The delete Operator (Page 2)**

- Formats:
  delete *object.propertyName*
  delete *object*["propertyName"]
- Example:
  delete aCircle.radius;
  delete aCircle["radius"];

32 **Try It Out**

- objects7.htm

33 **Testing Properties**

- The hasOwnProperty() method is a boolean method that returns a value indicating if a property exists in that object
- Format:
  object.hasOwnProperty("property")
- Example:
  document.write(aCircle.hasOwnProperty("x"));

34 **Try It Out**

- objects8.htm

35 **Enumerating through Properties (Page 1)**

- The for...in loop can be used to iterate through the properties of an object
- The *index* reference returns the *name* of the property as a string

36 **Enumerating through Properties (Page 2)**

- Format:
  for (*index* in *object*)
- Example:

```
for (index in aCircle)
{
    document.write(index);
}
```

37 **Try It Out**

- objects9.htm

38 **Enumerating through Properties (Page 3)**

- Since for...in returns the *property name* as type string ...
- It can be used in the *array notation* version of the property *value* access

39 **Enumerating through Properties (Page 4)**

- Format:
  object[*index*]
- Example:

```
for (index in aCircle)
{
   document.write( aCircle[index] );
}
```

40 **Try It Out**

- objects9a.htm

41 **Getters and Setters (Page 1)**

- A getter is a method that gets (accessed) the value of a specific property
- A setter is a method that sets (updates) the value of a specific property
- Getters and setters can be defined on any predefined core object or user-defined object that supports the addition of new properties
- The syntax for defining getters and setters uses the object literal syntax.

42 **Try It Out**

- objects10.htm

43 **The toString() Method**

44 **Try It Out**

- objects11.htm