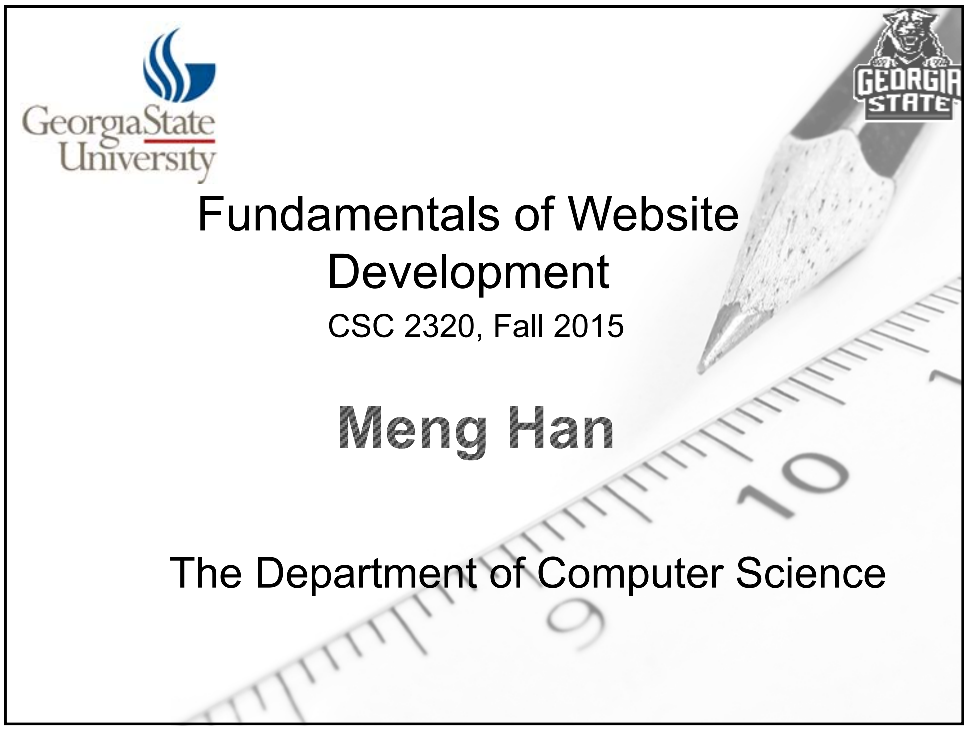
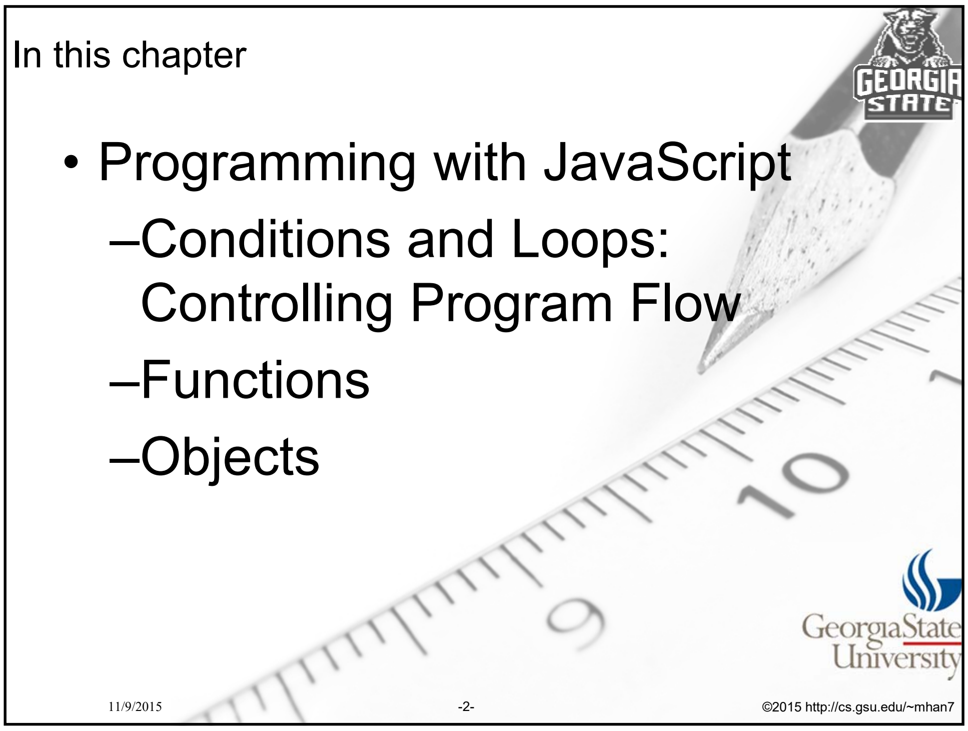


Fundamentals of Website Development

CSC 2320, Fall 2015

Meng Han

The Department of Computer Science

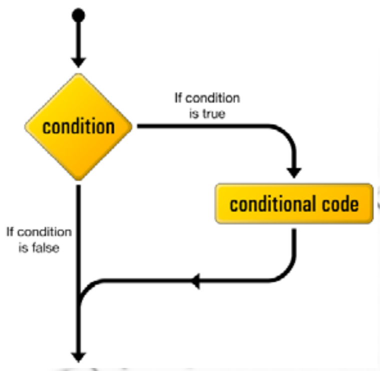


In this chapter

- Programming with JavaScript
  - –Conditions and Loops: Controlling Program Flow
  - –Functions
  - –Objects

Conditions

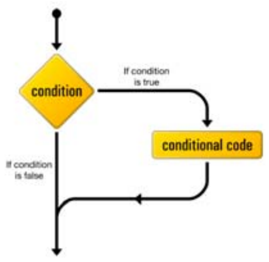
- If Statements
- Flow chart for If statement:



```
if (condition)
{
  conditional code;
}
```

Conditions(cont.)

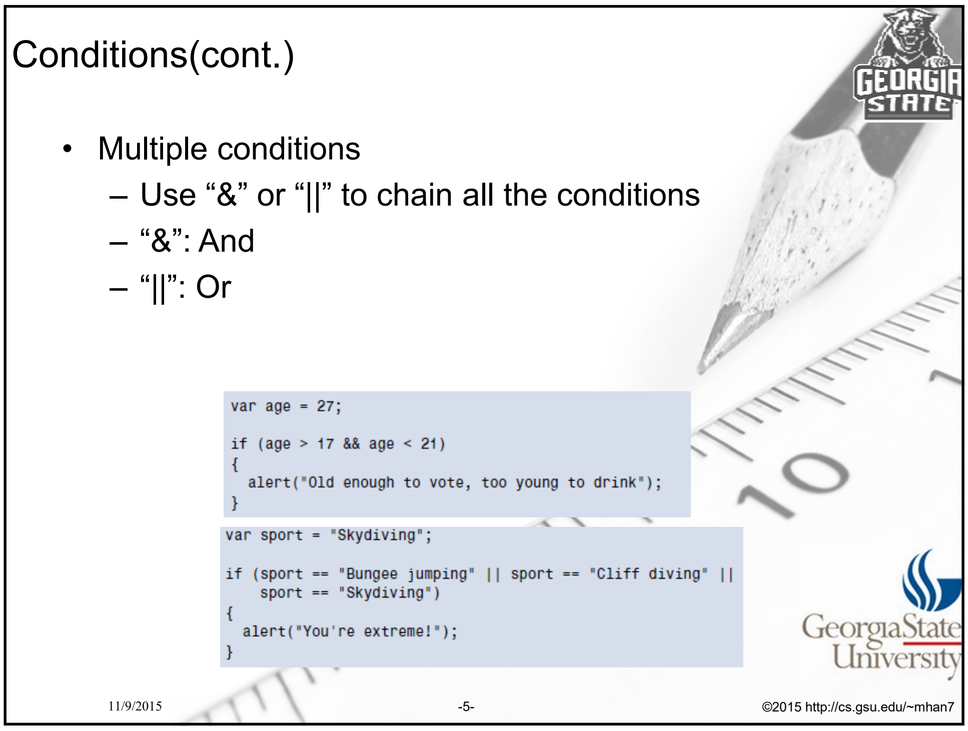
- E.g.



```
var age = 27;

if (age > 20)
{
  alert("Drink to get drunk");
}
```

```
var age = 27;

if (age == 50){
  alert("Half century");
}
```

```
var name = "Maximus";

if (name == "Maximus")
{
  alert("Good afternoon, General.");
}
```

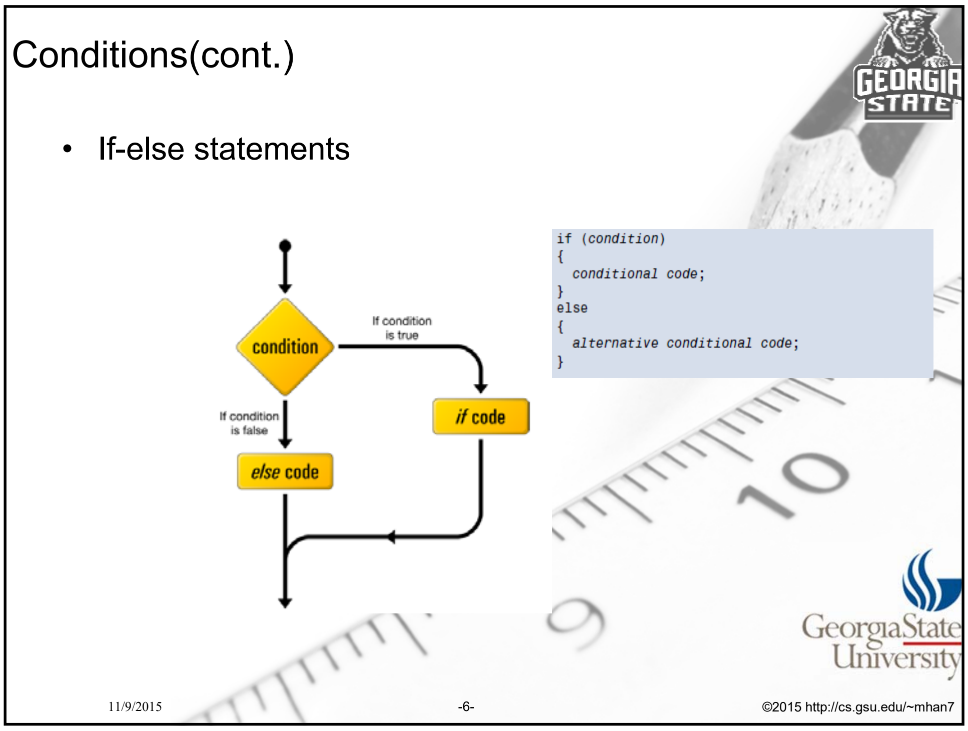


Conditions(cont.)

- Multiple conditions
  - Use “&” or “||” to chain all the conditions
  - “&”: And
  - “||”: Or

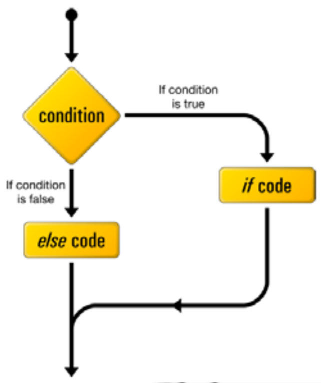
```
var age = 27;

if (age > 17 && age < 21)
{
  alert("Old enough to vote, too young to drink");
}
```

```
var sport = "Skydiving";

if (sport == "Bungee jumping" || sport == "Cliff diving" ||
    sport == "Skydiving")
{
  alert("You're extreme!");
}
```



Conditions(cont.)

- If-else statements



```
if (condition)
{
  conditional code;
}
else
{
  alternative conditional code;
}
```

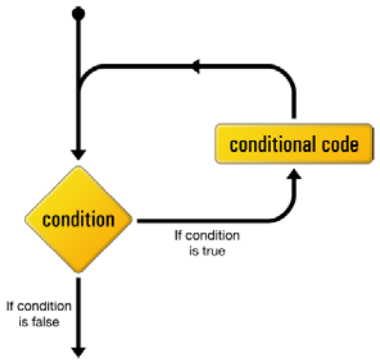
Conditions(cont.)

- Else-if statements
- E.g.

```
var name = "Marcus";

if (name == "Maximus")
{
  alert("Good afternoon, General.");
}
else if (name == "Marcus")
{
  alert("Good afternoon, Emperor.");
}
else
{
  alert("You are not allowed in.");
}
```

Loops

- While loops

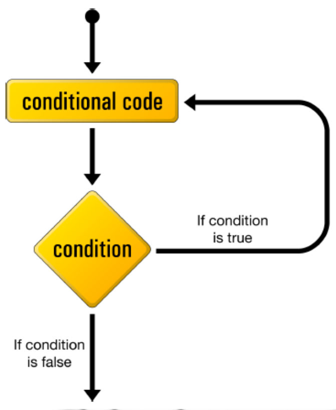


```
while (condition)
{
  conditional code;
}
```

```
var numbers = [1, 2, 3, 4, 5];
var incrementer = 0;
while (incrementer < numbers.length)
{
  numbers[incrementer] *= 2;
  incrementer++;
}
```

Loops(cont.)

- Do-while loops



```
do
{
  conditional code;
}
while (condition);
```

Loops(cont.)

- For loops
  - Most popular
  - Control structure is the same as while loops but more succinct

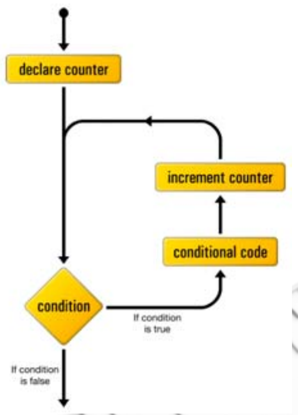
```
var numbers = [1, 2, 3, 4, 5];
var i = 0;
while (i < numbers.length)
{
  numbers[i] *= 2;
  i++;
}
```

```
var numbers = [1, 2, 3, 4, 5];

for (var i = 0; i < numbers.length; i++)
{
  numbers[i] *= 2;
}
```

  - Succinct in?
    - Initialization of the counter
    - The variation of the counter

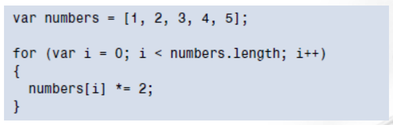
Loops(cont.)

- For loops



Loops(cont.)

- How to stop a loop halfway?
  - Use key word “break”;
- E.g.
  - How to break this after first three repetitions.



```
var numbers = [1, 2, 3, 4, 5];

for (var i = 0; i < numbers.length; i++)
{
  numbers[i] *= 2;
}
```

  - One way:
    - Add *if(i==3){ break;}* inside under the only statement.

Functions

- Question?
  - What if I want to re-run some code a few times?
  - E.g., alert function “alert()”;
- Function: Wrapper for a block of code
  - All you need is put a name for the wrapper
- Define function using key word: “*function*”

```
function warning()
{
  alert("This is your final warning");
}
```

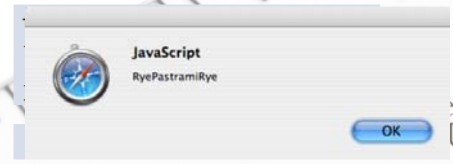
Functions(cont.)

- Arguments: pass data to a function
- E.g.

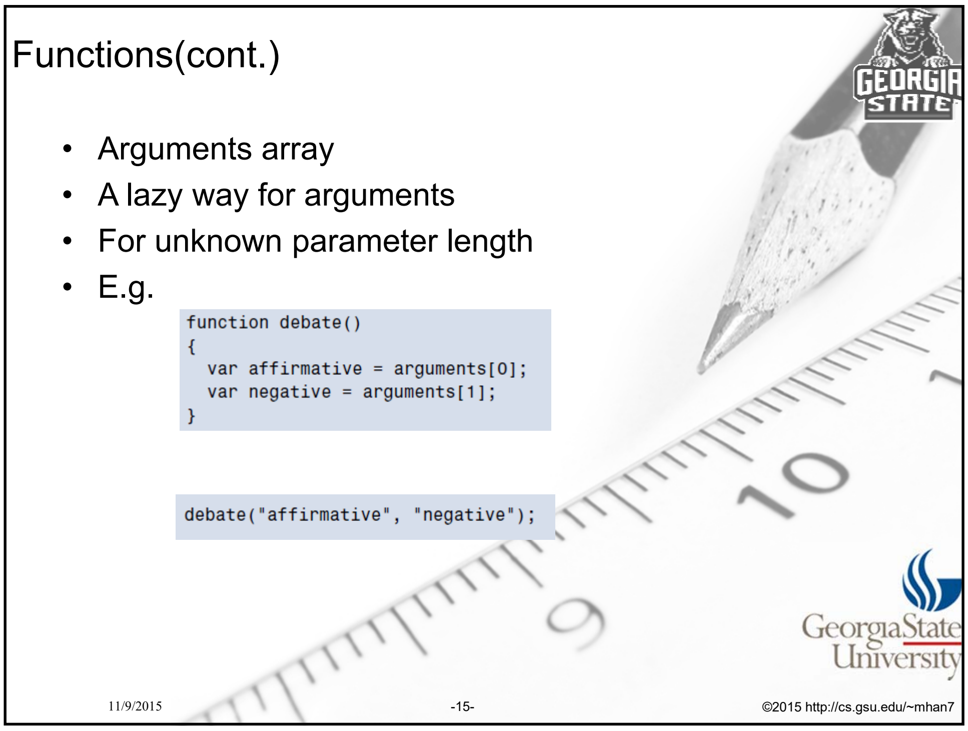
```
alert("This is your final warning");
```

```
function warning()
{
  alert("This is your final warning");
}
```

- E.g.
  - Definition:
  - Call:



JavaScript
RyePastramiRye



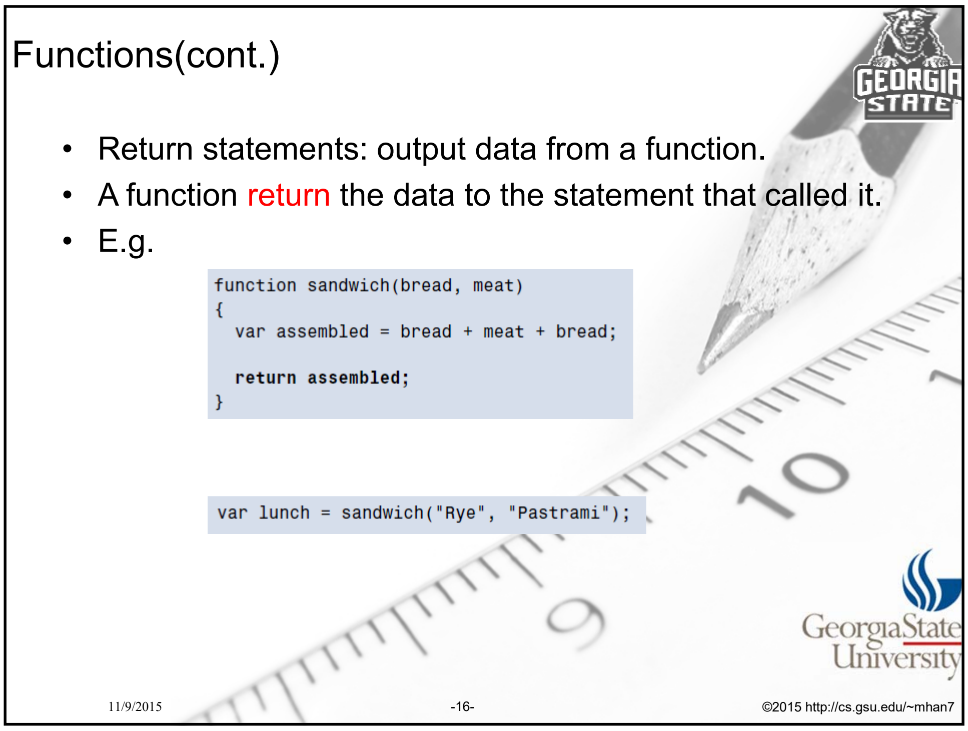
Functions(cont.)

- Arguments array
- A lazy way for arguments
- For unknown parameter length
- E.g.

```
function debate()
{
  var affirmative = arguments[0];
  var negative = arguments[1];
}
```

```
debate("affirmative", "negative");
```

11/9/2015 -15- ©2015 http://cs.gsu.edu/~mhan7



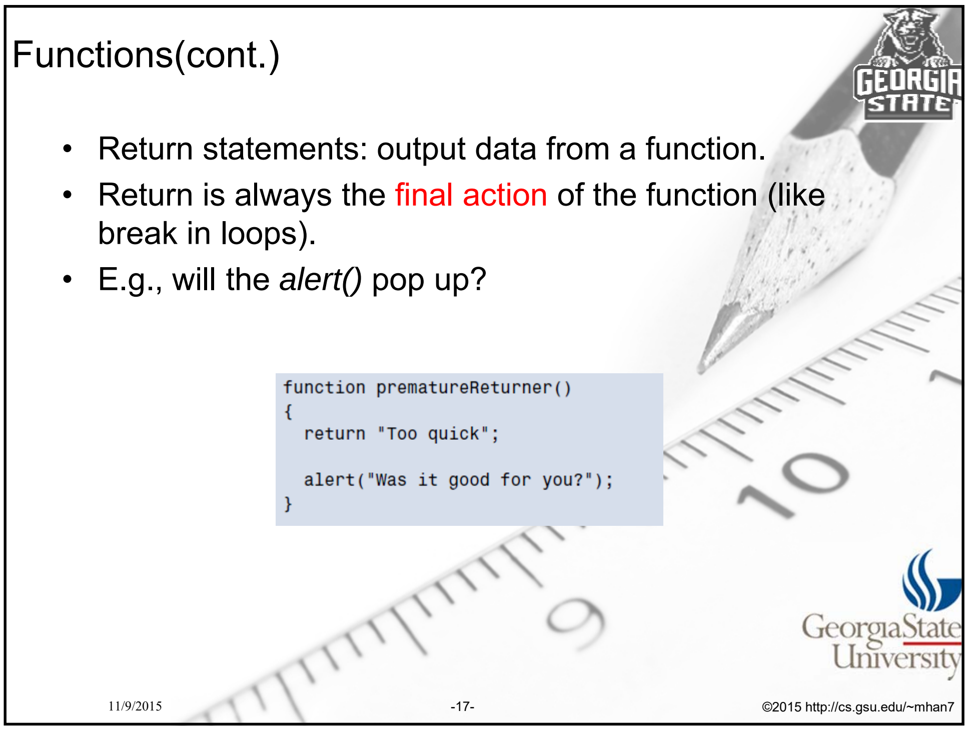
Functions(cont.)

- Return statements: output data from a function.
- A function return the data to the statement that called it.
- E.g.

```
function sandwich(bread, meat)
{
  var assembled = bread + meat + bread;

  return assembled;
}
```

```
var lunch = sandwich("Rye", "Pastrami");
```

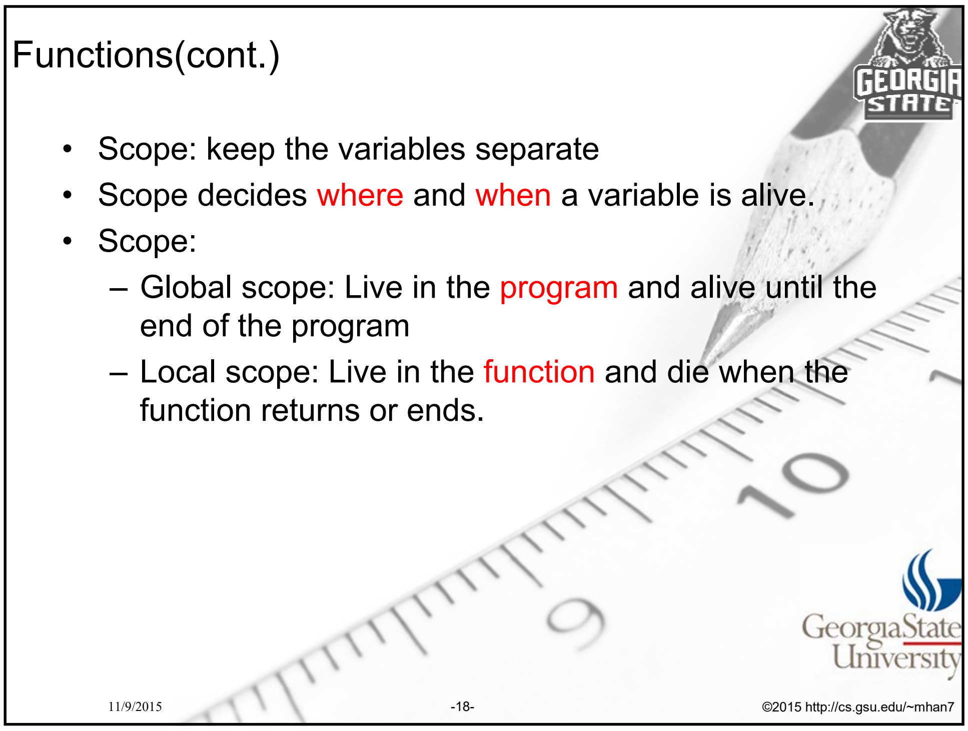
11/9/2015 -16- ©2015 http://cs.gsu.edu/~mhan7



Functions(cont.)

- Return statements: output data from a function.
- Return is always the final action of the function (like break in loops).
- E.g., will the *alert()* pop up?

```
function prematureReturner()
{
  return "Too quick";

  alert("Was it good for you?");
}
```



Functions(cont.)

- Scope: keep the variables separate
- Scope decides where and when a variable is alive.
- Scope:
  - Global scope: Live in the program and alive until the end of the program
  - Local scope: Live in the function and die when the function returns or ends.

Functions(cont.)

- E.g. for scope
- Name collision:
  - Stand for where it lives
  - E.g.

```
function countWiis()
{
  var stock = 5;
  var sales = 3;

  return stock - sales;
}

var stock = 0;
var wiis = countWiis();
```

Objects

- Objects: Group together sets of properties and methods.
  - Property: variable
  - Method: function
- An array is a native object
- Create your own objects

```
var Robot = new Object();
```

Objects(cont.)

- E.g., imagine the object as an array and its content may store native or complex data types or functions.

```
var Robot = new Object();

Robot.metal = "Titanium";
Robot.killAllHumans = function()
{
  alert("Exterminate!");
};

Robot.killAllHumans();
```

```
var Robot =
{
  metal: "Titanium",
  killAllHumans: function()
  {
    alert("Exterminate!");
  }
};
```

Homework

- Assignment 4
- Search numbers in your JavaScript.
  - Declare an array with the a sequence of numbers. E.g., following numbers in order. [9,3,4.3,24,54,8,19,23,46,87,3.14].
  - Task 1, pop up a window to output the index of number “23”.
  - Task 2, pop up a single window to output all the numbers that are larger than 10.
  - Alternatives: You better use functions to do the jobs. Like define a compare() function to test whether the current data is satisfied or not.
  - Due date: Nov. 18th, 2015.

Questions

Thank You!

Email: mhan@cs.gsu.edu

11/9/2015 -23- ©2015 http://cs.gsu.edu/~mhan7