MALMÖ UNIVERSITY

INLEDANDE WEBBPROGRAMMERING MED JAVASCRIPT

INTRODUCTION TO WEB PROGRAMING USING JAVASCRIPT

ME152A
L2: PROGRAM STRUCTURE & FUNCTIONS

OUTLINE

- What did we learn so far?
- Program structure
  - if else while do for loops
  - Functions
    - Function syntax
    - Parameters
    - Scopes
    - Local and global variables
  - Exercises during the class

WHAT DID WE LEARN SO FAR?

- Which are the primary types?
- What are the different types of operators?
- What is a vector?

PROGRAM STRUCTURE

- **Expressions and statements**
- Every value that is written literally (such as 22 or "psychoanalysis") is an expression.
- A program is simply a list of statements.
- The simplest kind of statement is an expression with a semicolon after it. This is a program:

```
1;
!false;
```

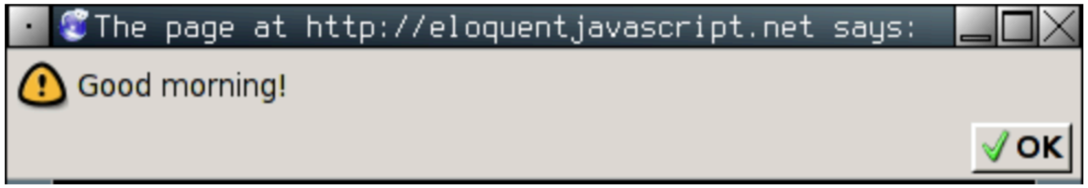
KEYWORDS AND RESERVED WORDS

```
break case catch class const continue debugger
default delete do else enum export extends false
finally for function if implements import in
instanceof interface let new null package private
protected public return static super switch this
throw true try typeof var void while with yield
```

THE ENVIRONMENT AND FUNCTIONS

- The collection of variables and their values that exist at a given time is called the *environment*.
- A lot of the values provided in the default environment have the type *function*.

```
alert("Good morning!");
```



The page at http://eloquentjavascript.net says:

Good morning!

THE CONSOLE.LOG FUNCTION AND RETURN VALUES

- The alert function is useful as an output device when experimenting
- Most JavaScript systems (including all modern web browsers and Node.js) provide a console.log function that writes out its arguments to *some* text output device.

```
var x = 30;
console.log("the value of x is", x);
// → the value of x is 30
```

- the function Math.max takes any number of number values and gives back the greatest.

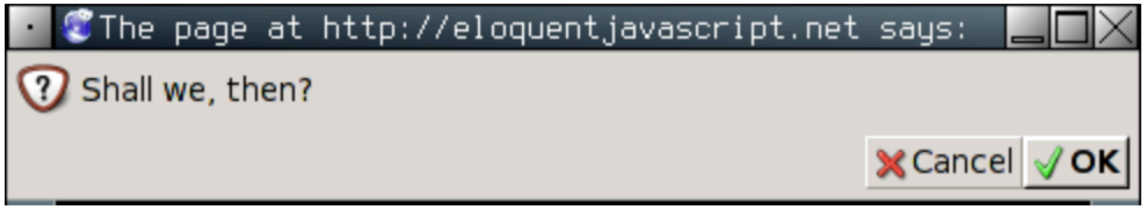
```
console.log(Math.max(2, 4));
// → 4
```

```
console.log(Math.min(2, 4) + 100);
// → 102
```

PROMPT AND CONFIRM

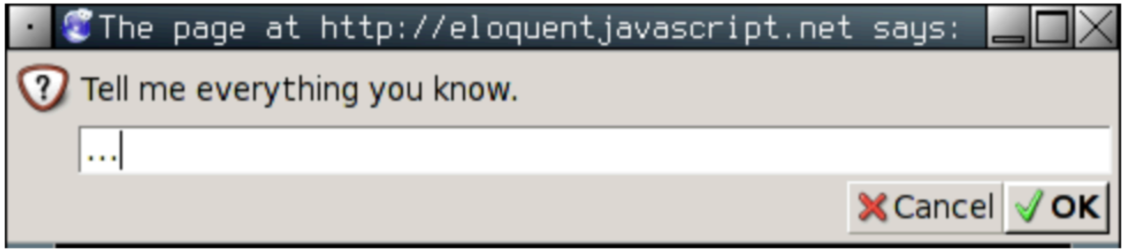
Browser environments contain other functions besides alert for popping up windows.

```
confirm("Shall we, then?");
```


The page at http://eloquentjavascript.net says:

Shall we, then?

```
prompt("Tell me everything you know.", "...");
```


The page at http://eloquentjavascript.net says:

Tell me everything you know.

...

CONTROL FLOW

When your program contains more than one statement, the statements are executed, predictably, from top to bottom.

```
var theNumber = Number(prompt("Pick a number", ""));
alert("Your number is the square root of " +
      theNumber * theNumber);
```

CONDITIONAL EXECUTION

We choose between two different routes based on a Boolean value - with `if` keyword:

```
var theNumber = Number(prompt("Pick a number", ""));
if (!isNaN(theNumber))
  alert("Your number is the square root of " +
        theNumber * theNumber);
```

with `if ... else` keyword:

```
var theNumber = Number(prompt("Pick a number", ""));
if (!isNaN(theNumber))
  alert("Your number is the square root of " +
        theNumber * theNumber);
else
  alert("Hey. Why didn't you give me a number?");
```

CONDITIONAL EXECUTION (CONT'D)

Example with more than two paths to choose from, multiple `if/else` pairs can be "chained" together.

```
var num = Number(prompt("Pick a number", "0"));

if (num < 10)
  alert("Small");
else if (num < 100)
  alert("Medium");
else
  alert("Large");
```

IF – ELSE EXERCISE

If the hour is less than 17, create a "Good day" greeting, otherwise "Good evening":

WHILE AND DO LOOPS

Looping control flow allows us to go back to some point in the program where we were before and repeat it with our current program state

```
console.log(0);
console.log(2);
console.log(4);
console.log(6);
console.log(8);
console.log(10);
console.log(12);
```

One way- not good

```
var number = 0;
while (number <= 12) {
  console.log(number);
  number = number + 2;
}
// → 0
// → 2
//   … etcetera
```

The other way

WHILE AND DO LOOPS (CONT'D)

Program that calculates and shows the value of 2^{10} (2 to the 10th power):

```
var result = 1;
var counter = 0;
while (counter < 10) {
  result = result * 2;
  counter = counter + 1;
}
console.log(result);
// → 1024
```

WHILE AND DO LOOPS (CONT'D)

The do loop is a control structure similar to the while loop. It differs only on one point: a do loop always executes its body at least once

```
do {
  var yourName = prompt("Who are you?");
} while (!yourName);
console.log(yourName);
```

LOOP WHILE EXERCISE

Make the loop start counting from 5. Count up to (including) 50 and count only every fifth number.

Declare a counter variable i.

Hint: set counter variable to 5, run the loop as long as i is less than or equal to 50, and i = i + 5.

FOR LOOPS

even-number-printing example:

```
for (var number = 0; number <= 12; number = number + 2)
  console.log(number);
// → 0
// → 2
//   … etcetera
```

code that computes 2^{10}:

```
var result = 1;
for (var counter = 0; counter < 10; counter = counter + 1)
  result = result * 2;
console.log(result);
// → 1024
```

BREAKING OUT OF A LOOP

A special statement called `break` has the effect of immediately jumping out of the enclosing loop.

```
for (var current = 20; ; current++) {
  if (current % 7 == 0)
    break;
}
console.log(current);
// → 21
```

DISPATCHING ON A VALUE WITH SWITCH

```
switch (prompt("What is the weather like?")) {
  case "rainy":
    console.log("Remember to bring an umbrella.");
    break;
  case "sunny":
    console.log("Dress lightly.");
  case "cloudy":
    console.log("Go outside.");
    break;
  default:
    console.log("Unknown weather type!");
    break;
}
```

REFLECTION

- A program is built out of statements
- Statements sometimes contain statements themselves
- Statements contain expressions – themselves contain smaller expressions
- Conditional statements:
  - `if`, `else`, and `switch`
- Looping statements
  - `while`, `do`, and `for`

FUNCTIONS

```
function myFunction(a, b)
```

DEFINING A FUNCTION

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).
- A function definition is just a regular variable definition where the value given to the variable happens to be a function.

```
var square = function(x) {
  return x * x;
};

console.log(square(12));
// → 144
```

http://www.w3schools.com/js/js_functions.asp

WHY FUNCTIONS?

- You can reuse code: Define the code once, and use it many times.

- You can use the same code many times with different arguments, to produce different results.

http://www.w3schools.com/js/js_functions.asp

KEEP IN MIND

```
var square = function(x) {
  return x * x;
};

console.log(square(12));
// → 144
```

- A function is created by an expression that starts with the keyword function.
- Functions have a set of *parameters* (in this case, only x) and a *body*, which contains the statements that are to be executed when the function is called.
- The function body must always be wrapped in braces, even when it consists of only a single statement
- A function can have multiple parameters or no parameters at all.

JAVASCRIPT FUNCTION SYNTAX

- A JavaScript function is defined with the **function** keyword, followed by a **name**, followed by parentheses **()**.
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas: ***(parameter1, parameter2, ...)***
- The code to be executed, by the function, is placed inside curly brackets: **{}**

```
function name(parameter1, parameter2, parameter3) {
    code to be executed
}
```

http://www.w3schools.com/js/js_functions.asp

EXAMPLE WITH NON AND TWO PARAMETERS

```
var makeNoise = function() {
  console.log("Pling!");
};

makeNoise();
// → Pling!

var power = function(base, exponent) {
  var result = 1;
  for (var count = 0; count < exponent; count++)
    result *= base;
  return result;
};

console.log(power(2, 10));
// → 1024
```

PARAMETERS AND SCOPES

- The parameters to a function behave like regular variables, but their initial values are given by the *caller* of the function, not the code in the function itself.
- An important property of functions is that the variables created inside of them, including their parameters, are *local* to the function.
- This “localness” of variables applies only to the parameters and to variables declared with the var keyword inside the function body.
- Variables declared outside of any function are called *global*, because they are visible throughout the program.

PARAMETER AND ARGUMENT

Argument

```
var square = function(x) {
    return x * x;
};

square(5); // 25
square(5, 10, 15, 25); // 25
square(); // NaN
```

Parameter

PARAMETERS AND SCOPES (CONT'D)

```
var x = "outside";

var f1 = function() {
  var x = "inside f1";
};
f1();
console.log(x);
// → outside

var f2 = function() {
  x = "inside f2";
};
f2();
console.log(x);
// → inside f2
```

FUNCTION INVOCATION

- The code inside the function will execute when "something" **invokes** (calls) the function:
  - When an event occurs (when a user clicks a button)
  - When it is invoked (called) from JavaScript code
  - Automatically (self invoked)

http://www.w3schools.com/js/js_functions.asp

FUNCTION RETURN

- When JavaScript reaches a **return statement**, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- Functions often compute a **return value**. The return value is "returned" back to the "caller":

http://www.w3schools.com/js/js_functions.asp

JAVASCRIPT SCOPE

- Objects and functions are also variables.
- Scope is the set of variables, objects, and functions you have access to.
- JavaScript has function scope: The scope changes inside functions.

http://www.w3schools.com/js/js_scope.asp

LOCAL VARIABLES

- Variables declared within a JavaScript function, become **LOCAL** to the function.
- Local variables have **local scope**: They can only be accessed within the function.

```
// code here can not use carName

function myFunction() {
    var carName = "Volvo";

    // code here can use carName

}
```

http://www.w3schools.com/js/js_scope.asp

GLOBAL VARIABLES

- A variable declared outside a function, becomes **GLOBAL**.
- A global variable has **global scope**: All scripts and functions on a web page can access it.

```
var carName = "Volvo";

// code here can use carName

function myFunction() {

    // code here can use carName

}
```

http://www.w3schools.com/js/js_scope.asp

AUTOMATICALLY GLOBAL

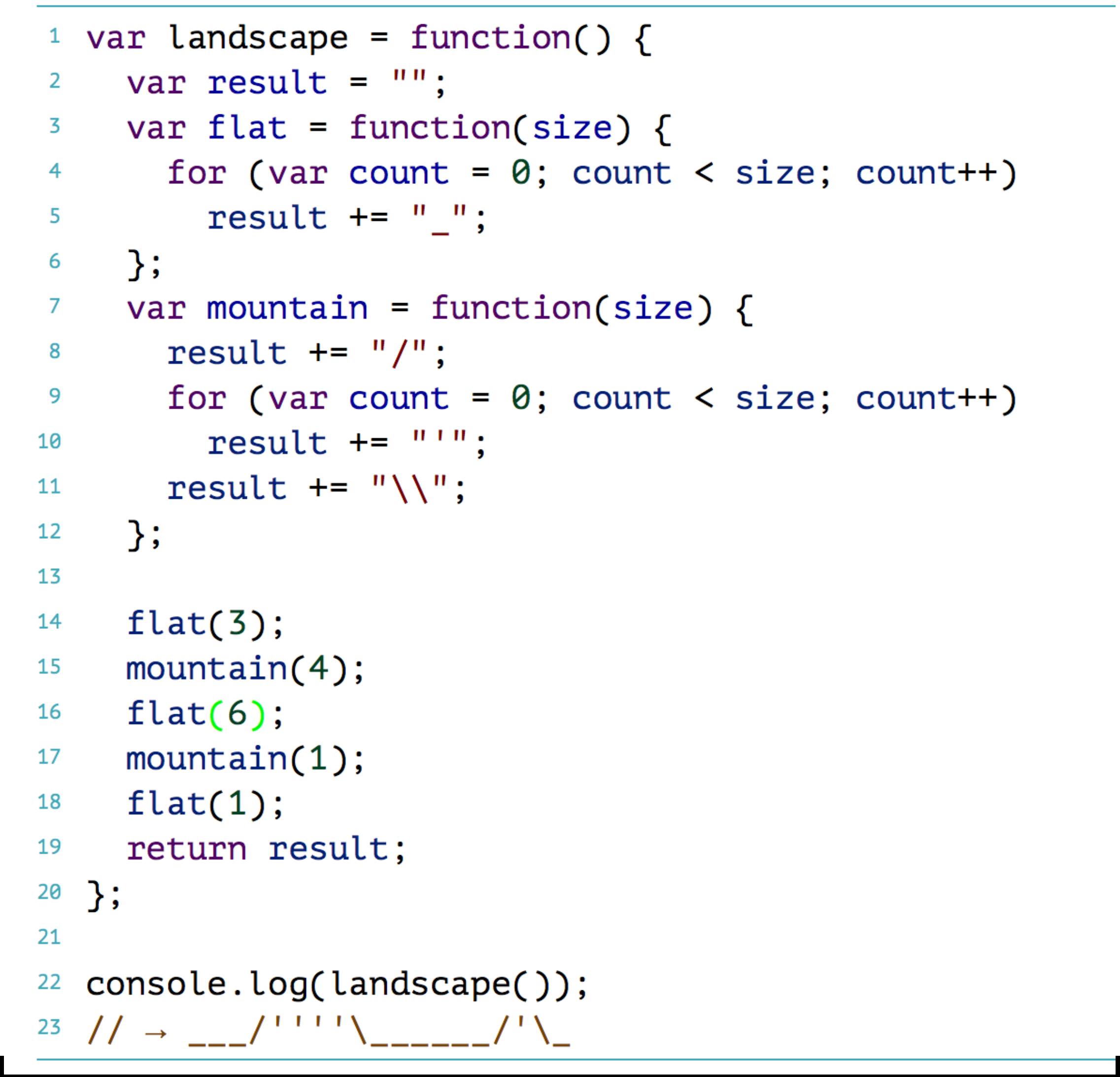
- If you assign a value to a variable that has not been declared, it will automatically become a **GLOBAL** variable.

- This code example will declare carName as a global variable, even if it is executed inside a function.

```
// code here can use carName

function myFunction() {
    carName = "Volvo";

    // code here can use carName

}
```

http://www.w3schools.com/js/js_scope.asp

NESTED SCOPE



```
var landscape = function() {
  var result = "";
  var flat = function(size) {
    for (var count = 0; count < size; count++)
      result += "_";
  };
  var mountain = function(size) {
    result += "/";
    for (var count = 0; count < size; count++)
      result += "'";
    result += "\\";
  };

  flat(3);
  mountain(4);
  flat(6);
  mountain(1);
  flat(1);
  return result;
};

console.log(landscape());
// → ___/''''\______/'\_
```

THE CALL STACK

- Control flows through functions:

```
function greet(who) {
  console.log("Hello " + who);
}
greet("Harry");
console.log("Bye");
```

Schematically:

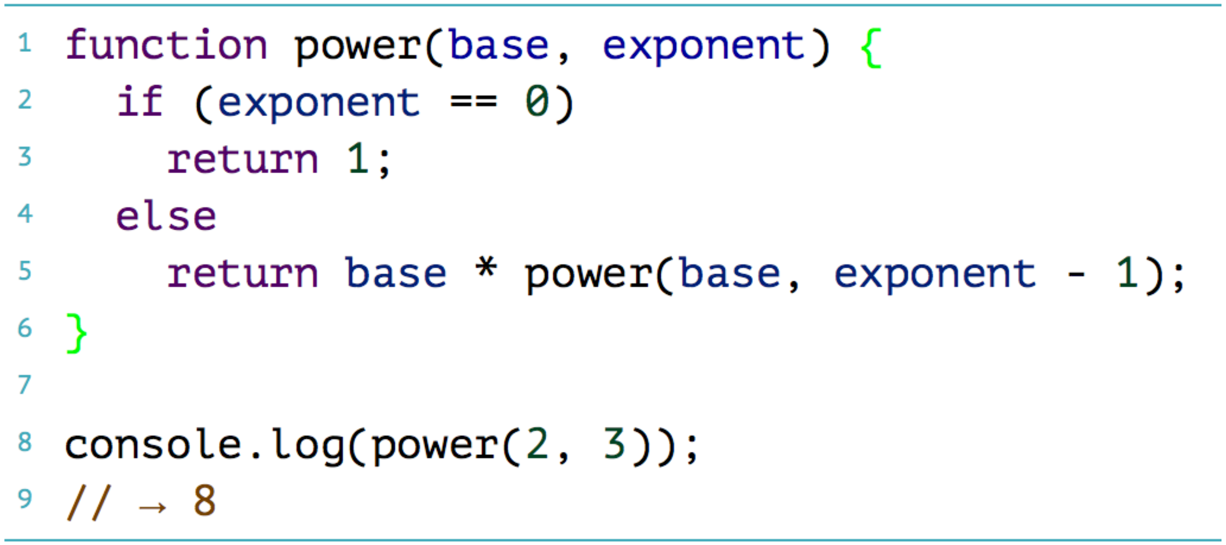
```
top
   greet
        console.log
   greet
top
   console.log
top
```

out of stack space:

```
function chicken() {
  return egg();
}
function egg() {
  return chicken();
}
console.log(chicken() + " came first.");
// → ??
```

RECURSION

- A function that calls itself is called *recursive*
- Recursion allows some functions to be written in a different style.



```
function power(base, exponent) {
  if (exponent == 0)
    return 1;
  else
    return base * power(base, exponent - 1);
}

console.log(power(2, 3));
// → 8
```

- Recursion is a technique for iterating over an operation by having a function call itself repeatedly until it arrives at a result.
- Recursion is best applied when you need to call the same function repeatedly with different parameters from within a loop.

http://www.sitepoint.com/recursion-functional-javascript/

FUNCTION EXERCISE: CALCULATE YOUR AGE

- Write a function named calAge:
  - 2 arguments: birth year, current year.
  - calculate 2 possible ages.
  - outputs the result: "You are NN"
- Call the function three times with different sets of values.

BILL GATES EXPLAINING PROGRAMMING (WRITING CODES)

https://www.youtube.com/watch?v=BWbh3hcK0A0

REFLECTION

- The function keyword, when used as an expression, can create a function value.
- When used as a statement, it can be used to declare a variable and give it a function as its value.
- A key aspect in understanding functions is understanding local scopes.
- Parameters and variables declared inside a function are local to the function, re-created every time the function is called, and not visible from the outside.
- Functions can make a program more readable by grouping code into conceptual chunks

```javascript
// Create a function value f
var f = function(a) {
  console.log(a + 2);
};

// Declare g to be a function
function g(a, b) {
  return a * b * 3.5;
}
```

THANK YOU

QUESTIONS?

Literature:

Haverbeke, M. (2014). *Eloquent JavaScript: A Modern Introduction to Programming*. No Starch Press.