Exercises: Basic

For all of these exercises, use the “more powerful practice” approach from the first lecture, where you load an HTML file that has a script tag pointing at a JavaScript file, test the function calls in Firebug or Chrome, and reload the HTML page each time you change the function definitions.

1. Make a function that returns “even” or “odd” depending on the number passed to it.

```
parity(1); --> "odd"
parity(2); --> "even"
```

2. The notes showed a simple version of max that took exactly two arguments. Update this to take exactly three arguments. Note that the builtin version of Math.max takes any number of arguments, which is much better, but we don’t know how to do variable arguments yet. And, of course, no using Math.max on this exercise.

```
max(1, 2, 3); --> 3
max(1, 3, 2); --> 3
max(3, 2, 1); --> 3
```

3. Copy the flipCoin function from the last set of exercises. Now, make a function that, given a number, flips a coin that many times and returns the number of heads.

```
numHeads(10); --> 4
numHeads(10); --> 6
numHeads(10); --> 6
```

4. Make a function that takes a number of flips and returns the fraction that were heads.

```
headsRatio(10); --> 0.7
headsRatio(10); --> 0.4
headsRatio(10000); --> 0.5023
headsRatio(10000000); --> 0.4999948
```

5. Make a function that takes a number and a short string, and returns the string concatenated that number of times.

```
padChars(5, "x"); --> "xxxxx"
padChars(7, "-"); --> "-------"
```

6. Write a function that returns the number of times you have to roll a die to get a 6.
(Minor hint: compare Math.random() to 5/6).

```
numRollsToGetSix(); --> 13
numRollsToGetSix(); --> 2
```

7. Update the HTML page so that each time you reload it, you randomly see either a “Have a GOOD day!” or “Have a BAD day!” message. If you know some CSS already, make the font big.

Exercises: Basic JS

The last two problems are quite a bit more difficult than the first three.

1. Create an array containing four random numbers. Use one-step array allocation. Print out the array by evaluating the array variable in Firebug.

```
var fourNums = ...;
fourNums; --> [0.871570877817405, 0.9107447521970577,
               0.743357509580703, 0.6571292972456975]
```

2. Create an array containing 100 random numbers. Use two-step array allocation. Print out the array.

```
var hundredNums = ...;
for(...) { ... }
hundredNums; --> [0.8742489161574934, 0.7147785711684753, 0.8062322101495641,
                  ...
                 0.41288219216760613, 0.5113443687277072]
```

3. Make a function that given an array of strings, where each string represents a number, returns an array of the corresponding numbers.

```
var strings = ["1.2", "2.3", "3.4"];
var nums = numberArray(strings);
nums; --> [1.2, 2.3, 3.4]
```

4. Write a function that, given a string, will return the longest token (consecutive string of characters) that contains neither an a nor a b.

```
longestToken("ababcdababefgababhiab"); --> "efg"
longestToken("aba"); --> ""
```

5. Write a function that, given an array of strings, will compute the sum of the lengths of the words that do not contain a “q”. Do not use a loop or the forEach method, only array methods (filter, map, reduce) and string methods/properties (indexOf, length).

```
var test1 = ["stop", "quit", "exit"];
lengthOfNonQWords(test1); --> 8
var test2 = ["queen", "quit"];
lengthOfNonQWords(test2); --> 0
```

Exercises - Basic JS

1. Try JavaScript interactively

   - If you use Firefox, install Firebug from http://getfirebug.com/. Copy the simple test-page.html from the section's source code archive (javascript-getting-started), load it in Firefox, then launch Firebug by clicking on the Firebug icon or hitting F12. Click on the Console tab and try some interactive JavaScript commands. Customize the console as described in the notes if you wish.
   - If you use Chrome, copy the simple test-page.html from the section's source code archive (javascript-getting-started), load it in Chrome, then launch the developer tools with Control-Shift-J. Click on the Console tab and try some interactive JavaScript commands. Most browsers, even Internet Explorer and Microsoft Edge have something similar now.

2. Define a variable x and give it a value of 5. Evaluate x and verify it shows the value.

3. Enter this function into the console:

```
function half(x) {
  return(x / 2);
}
```

   Try to predict what you will get for half(x), half(4), and half(3). It is simple to predict what you will get for half(4), but, depending on your programming background, it might not be so easy to predict what you will get for half(x) and half(3). Call half(x), half(4), and half(3) and see if you were right.

4. Try to predict what you will get if you evaluate x in the console. Is it still 5, or is it 3? Try it and see.

5. Enter this function into the console:

```
function seven() {
  x = 7;
  return(x);
}
```

   Call seven() in the console. Try to predict what you will get if you evaluate x in the console. Try it and see. How do you explain the surprising result?

6. Make a function called calculation that, given three numbers a, b, and c, returns (a + b)/c. Try it with a few normal values. Then, try to predict what you will get for calculation(1, 1, 0), calculation(-1, -1, 0), and calculation(1, -1, 0). Try those tests and see if you get what you expected.

7. Try the "more powerful practice" approach from the notes, where you load an HTML file that has a script tag pointing at a JavaScript file. Put a function definition in the JS file, load the HTML file, go to the console, and call the function. Then, add a second function definition to the JS file, reload the HTML file, return to the console, and call the new function.

8. Write a function called isEven that, given a number, returns true if the number is even and false if the number is odd. This would be simple if we had covered if statements or the ?: ternary operator, but we haven't, so you cannot use either one.

1) Write a program that prints ‘Hello World’ to the screen.
2) Write a program that asks the user for their name and greets them with their name.
3) Modify the previous program such that only the users Alice and Bob are greeted with their names.
4) Write a program that asks the user for a number n and prints the sum of the numbers 1 to n
5) Modify the previous program such that only multiples of three or five are considered in the sum, e.g. 3, 5, 6, 9, 10, 12, 15 for n=17
6) Write a program that asks the user for a number n and gives them the possibility to choose between computing the sum and computing the product of 1,…,n.
7) Write a program that prints a multiplication table for numbers up to 12.
8) Write a program that prints all prime numbers. (Note: if your programming language does not support arbitrary size numbers, printing all primes up to the largest number you can easily represent is fine too.)
9) Write a guessing game where the user has to guess a secret number. After every guess the program tells the user whether their number was too large or too small. At the end the number of tries needed should be printed. It counts only as one try if they input the same number multiple times consecutively.
10) Write a program that prints the next 20 leap years.
11) Write a function that returns the largest element in a list.
12) Write function that reverses a list, preferably in place.
13) Write a function that checks whether an element occurs in a list.
14) Write a function that returns the elements on odd positions in a list.
15) Write a function that computes the running total of a list.
16) Write a function that tests whether a string is a palindrome.
17) Write three functions that compute the sum of the numbers in a list: using a for-loop, a while-loop and recursion. (Subject to availability of these constructs in your language of choice.)
18) Write a function on_all that applies a function to every element of a list. Use it to print the first twenty perfect squares. The perfect squares can be found by multiplying each natural number with itself. The first few perfect squares are 1*1= 1, 2*2=4, 3*3=9, 4*4=16. Twelve for example is not a perfect square because there is no natural number m so that m*m=12. (This question is tricky if your programming language makes it difficult to pass functions as arguments.)
19) Write a function that concatenates two lists. [a,b,c], [1,2,3] → [a,b,c,1,2,3]
20) Write a function that combines two lists by alternatingly taking elements, e.g. [a,b,c], [1,2,3] → [a,1,b,2,c,3].
21) Write a function that merges two sorted lists into a new sorted list. [1,4,6],[2,3,5] → [1,2,3,4,5,6]. You can do this quicker than concatenating them followed by a sort.

22) Write a function that rotates a list by k elements. For example [1,2,3,4,5,6] rotated by two becomes [3,4,5,6,1,2]. Try solving this without creating a copy of the list. How many swap or move operations do you need?
23) Write a function that computes the list of the first 100 Fibonacci numbers. The first two Fibonacci numbers are 1 and 1. The n+1-st Fibonacci number can be computed by adding the n-th and the n-1-th Fibonacci number. The first few are therefore 1, 1, 1+1=2, 1+2=3, 2+3=5, 3+5=8.
24) Write a function that takes a number and returns a list of its digits. So for 2342 it should return [2,3,4,2].
25) Write functions that add, subtract, and multiply two numbers in their digit-list representation (and return a new digit list). If you're ambitious you can implement Karatsuba multiplication. Try different bases. What is the best base if you care about speed? If you couldn't completely solve the prime number exercise above due to the lack of large numbers in your language, you can now use your own library for this task.
26) Write a function that takes a list of numbers, a starting base b1 and a target base b2 and interprets the list as a number in base b1 and converts it into a number in base b2 (in the form of a list-of-digits). So for example [2,1,0] in base 3 gets converted to base 10 as [2,1].
27) Implement the following sorting algorithms: Selection sort, Insertion sort, Merge sort, Quick sort, Stooge Sort. Check Wikipedia for descriptions.
28) Implement binary search.
29) Write a function that takes a list of strings and prints them, one per line, in a rectangular frame. For example the list ["Hello", "World", "in", "a", "frame"] gets printed as:

```
*********
* Hello *
* World *
* in    *
* a     *
* frame *
*********
```

30) Write function that translates a text to Pig Latin and back. English is translated to Pig Latin by taking the first letter of every word, moving it to the end of the word and adding ‘ay’. “The quick brown fox” becomes “Hetay uickqay rownbay oxfay”.

Java Script Assignments

Assignment 1:

Create a simple page that lets users type in some temperature value in the Fahrenheit scale and when the user clicks a “Show results” button, to show the temperature in Celsius scale. For example, if the user types in 32, your results should show “0 degree Celsius.”

Functionality

At the least, your program should allow for the following

User Input: One text field where the user will type in the temperature in Fahrenheit

Input Validation: When the “Show results” button is clicked, your program should check to see if the user has left the text field empty and also if what the user typed is a number and not anything else.

If there are problems (as in II. above) then your program should show appropriate error messages

Display results: If there are no errors and user input is valid, then your program should show the results to the user.

Assignment 2:

Write a script which uses a prompt box to get an input. Validate that the input is an Integer between 1 and 30, and then print to the page asterisks (*) to represent the number. Your script must run until you have collected 3 valid inputs as well as generate 3 outputs (one output on one line).
Example:
Input: 4
Input: 6
Input: 12

```
****
******
************
```

Assignment 3:

Create an HTML page that will Prompt the user:

It should take input for the number of rows and the number of columns

Then it should create a table (HTML table) with the given number of row and columns

Each cell of the table should contain the cell id (row#, col#)

Assignment 4:

In this assignment, we will write a JavaScript web application called MyCalculator. All 3 files (the .js, .html and .css files should be in the MyCalculator folder and should be called MyCalculator.XXX).

The program should do the following:

-There should be one resultBox in the .html page and one runCalculator button. This should be clicked to run the program.

-Present a menu of 7 choices to the user, using a prompt:

1. Add

2. Subtract
3. Multiply
4. Divide
5. Exponent
6. Mean
7. Quit

- If the user selects anything other than 1-7, the program should print an error message and ask the user to select again. Use a for loop to check this.

-If the user selects 1, the program should ask the user to enter the first number and then the second, and then print out the result in the resultBox. Then it should display the prompt menu again.

-Similar behavior for choices 2,3 and 4: for subtract, subtract the second number from the first, for divide, divide the first number by the second.

-We should do isNAN checks for all numbers. Use a for loop to check that each input is a number.

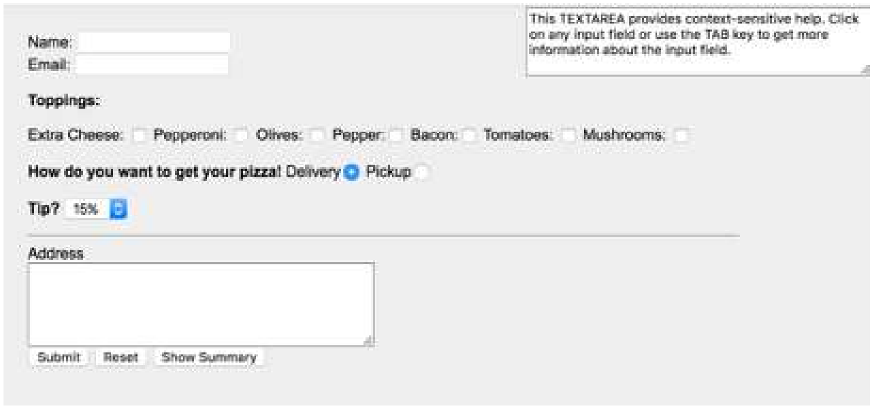
-For choice 5, the program should ask for the base, and then the exponent, and then print out the answer, using the Math.pow() method or equivalent.

-For choice 6, the program should prompt the user to input a series of numbers. The series ends, when the user enters a “***”. At this point, our program should print out the **mean** value, and then show the main menu prompt again.

Assignment 5:

In this assignment you will practice basic HTML, Javascript and CSS concepts

You are asked to build a page to order pizza! The page asks the user to enter information regarding the pizza order and then see a summary of what have been entered. The page is static HTML; however, part of the page will be dynamically built as it will be explained below.



This TEXTAREA provides context-sensitive help. Click on any input field or use the TAB key to get more information about the input field.

Name:
Email:

Toppings:

Extra Cheese: ☐ Pepperoni: ☐ Olives: ☐ Pepper: ☐ Bacon: ☐ Tomatoes: ☐ Mushrooms: ☐

How do you want to get your pizza! Delivery [✓ ●] Pickup ○

Tip? 15%

Address

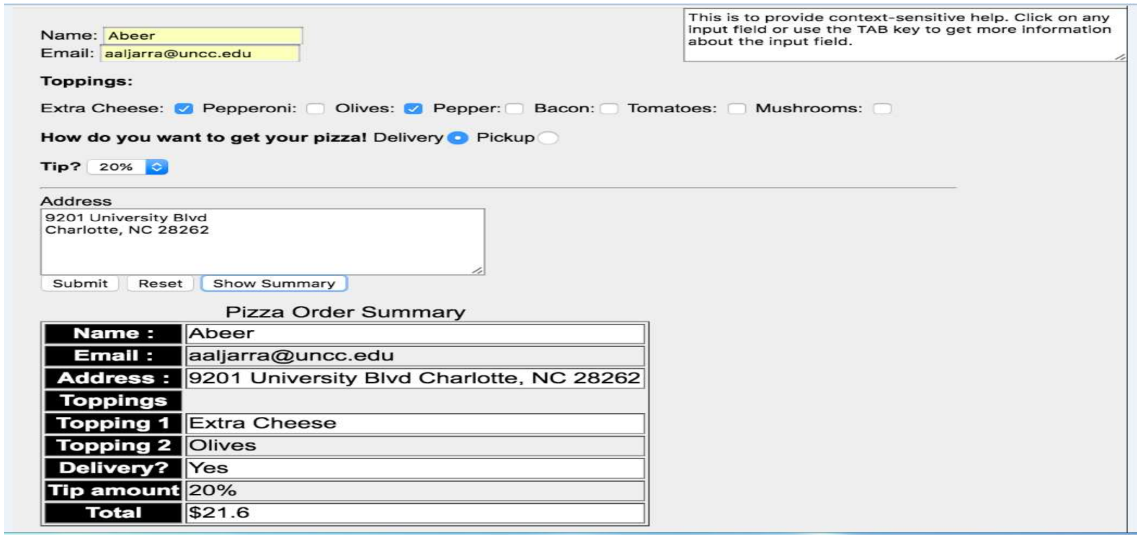
Figure 1

The page consists of the following fields:
1. Customer Name & Email (Input fields)
2. Toppings Options (Check Boxes)
3. Delivery method (Radio Buttons)
4. Tip amount (Drop Down Menu), values are 15%, 20%, and 25%
5. Address (Text Area)
6. Action Buttons
7. Help Text (Text Area)

8. Summary of Order (Table)
All the fields are static fields except the last one (Summary Table) which is built based on the values entered by the user.
CSS is used to apply the following visual properties:
• Input fields width is 600px
• Font type: arial
• Font size: 80%
• Background: light gray (#eee)

• Margin: 20px
• Table even rows: background color light gray (#eee)
• Table odd rows: background color white (#fff)
• Table header row: background color black, font color white

This page should implement the following actions (see Figure 2):
1. The Help Text (Field # 1 in Figure 2) is responsible of showing a Hint message to explaining information about the field that is currently focused (cursor is on the field). If there is no focus on any field, display a default hint message. Messages are displayed on (Table 1).
2. Submit Button should check that all fields are filled otherwise display an error message indicating that there exist missing values.
3. Clear All: Clear All fields.
4. Show Summary: This button is initially disabled. It is enables only when submit button is pressed and all fields values are valid. Clicking this button should build the Summary Table (see #2 in Figure 2). The table will summarize the pizza order based on what have been filled. The table should show:
• Customer Name
• Customer Email
• Customer Address
• Toppings selected
• Delivery option
• Total Price based on this formula: (base price + 1.5 * #of toppings + delivery fee)*1.0+tip. Where: base price= 10 and delivery is 5. For example, if 3 toppings are selected, delivery is NOT checked and 20% tip is selected, total is (10 + 1.5 * 3 + 0) * 1.2 = $17.4


Name: Abeer
Email: aaljarra@uncc.edu

This is to provide context-sensitive help. Click on any input field or use the TAB key to get more information about the input field.

Toppings:

Extra Cheese: ✓ Pepperoni: ☐ Olives: ✓ Pepper: ☐ Bacon: ☐ Tomatoes: ☐ Mushrooms: ☐

How do you want to get your pizza! Delivery ● Pickup ○

Tip? 20%

Address
9201 University Blvd
Charlotte, NC 28262

Pizza Order Summary

Name :	Abeer
Email :	aaljarra@uncc.edu
Address :	9201 University Blvd Charlotte, NC 28262
Toppings	
Topping 1	Extra Cheese
Topping 2	Olives
Delivery?	Yes
Tip amount	20%
Total	$21.6

Figure 2

1. Write a program that outputs all possibilities to put + or - or nothing between the numbers 1,2,…,9 (in this order) such that the result is 100. For example 1 + 2 + 3 - 4 + 5 + 6 + 78 + 9 = 100.
2. Write a program that takes the duration of a year (in fractional days) for an imaginary planet as an input and produces a leap-year rule that minimizes the difference to the planet's solar year.
3. Implement a data structure for graphs that allows modification (insertion, deletion). It should be possible to store values at edges and nodes. It might be easiest to use a dictionary of (node, edgelist) to do this.
4. Write a function that generates a DOT representation of a graph.
5. Write a program that automatically generates essays for you.
   a. Using a sample text, create a directed (multi-)graph where the words of a text are nodes and
      there is a directed edge between u and v if u is followed by v in your sample text. Multiple occurrences lead to multiple edges.
   b. Do a random walk on this graph: Starting from an arbitrary node choose a random successor.
      If no successor exists, choose another random node.
6. Write a program that automatically converts English text to Morse code and vice versa.
7. Write a program that finds the longest palindromic substring of a given string. Try to be as efficient as possible!
8. Think of a good interface for a list. What operations do you typically need? You might want to investigate the list interface in your language and in some other popular languages for inspiration.
9. Implement your list interface using a fixed chunk of memory, say an array of size 100. If the user wants to add more stuff to your list than fits in your memory you should produce some kind of error, for example you can throw an exception if your language supports that.
10. Improve your previous implementation such that an arbitrary number of elements can be stored in your list. You can for example allocate bigger and bigger chunks of memory as your list grows, copy the old elements over and release the old storage. You should probably also release this memory eventually if your list shrinks enough not to need it anymore. Think about how much bigger the new chunk of memory should be so that your performance won't be killed by allocations. Increasing the size by 1 element for example is a bad idea.
11. If you choose your growth right in the previous problem, you typically won't allocate very often. However, adding to a big list sometimes consumes considerable time. That might be problematic in some applications. Instead try allocating new chunks of memory for new items. So when your list is full and the user wants to add something, allocate a new chunk of 100 elements instead of copying all elements over to a new large chunk. Think about where to do the book-keeping about which chunks you have. Different book keeping strategies can quite dramatically change the performance characteristics of your list.

12. Implement a binary heap. Once using a list as the base data structure and once by implementing a pointer-linked binary tree. Use it for implementing heap-sort.
13. Implement an unbalanced binary search tree.
14. Implement a balanced binary search tree of your choice. I like (a,b)-trees best.
15. Compare the performance of insertion, deletion and search on your unbalanced search tree with your balanced search tree and a sorted list. Think about good input sequences. If you implemented an (a,b)-tree, think about good values of a and b.
16. Given two strings, write a program that efficiently finds the longest common subsequence.
17. Given an array with numbers, write a program that efficiently answers queries of the form: “Which is the nearest larger value for the number at position i?”, where distance is the difference in array indices. For example in the array [1,4,3,2,5,7], the nearest larger value for 4 is 5. After linear time preprocessing you should be able to answer queries in constant time.
18. Given two strings, write a program that outputs the shortest sequence of character insertions and deletions that turn one string into the other.
19. Write a function that multiplies two matrices together. Make it as efficient as you can and compare the performance to a polished linear algebra library for your language. You might want to read about Strassen’s algorithm and the effects CPU caches have. Try out different matrix layouts and see what happens.
20. Implement a van Emde Boas tree. Compare it with your previous search tree implementations.
21. Given a set of d-dimensional rectangular boxes, write a program that computes the volume of their union. Start with 2D and work your way up.